26th International Conference on Theory and Applications of Satisfiability Testing

SAT 2023, July 4–8, 2023, Alghero, Italy

Edited by

Meena Mahajan
Friedrich Slivovsky

LIPIcs

Editors

Meena Mahajan
The Institute of Mathematical Sciences, HBNI, Chennai, India
meena@imsc.res.in

Friedrich Slivovsky
TU Wien, Austria
fs@ac.tuwien.ac.at

ACM Classification 2012
Theory of computation → Automated reasoning; Theory of computation → Logic and verification; Hardware → Hardware validation

ISBN 978-3-95977-286-0

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/978-3-95977-286-0.

Publication date
August, 2023

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.SAT.2023.0

ISBN 978-3-95977-286-0 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Papers

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at SAT 2023, the 26th International Conference on Theory and Applications of Satisfiability Testing, held from July 4–8, 2023 in Alghero, Sardinia, Italy. The SAT 2023 conference was hosted by the University of Sassari in its Alghero campus.

The International Conference on Theory and Applications of Satisfiability Testing (SAT) is the premier annual meeting for researchers focusing on the theory and applications of the propositional satisfiability problem, broadly construed. In addition to plain propositional satisfiability, it also includes Boolean optimization (such as MaxSAT and Pseudo-Boolean (PB) constraints), Quantified Boolean Formulas (QBF), Satisfiability Modulo Theories (SMT), Model Counting, and Constraint Programming (CP) for problems with clear connections to Boolean-level reasoning. Many hard combinatorial problems can be tackled using SAT-based techniques including problems that arise in Formal Verification, Artificial Intelligence, Operations Research, Computational Biology, Cryptography, Data Mining, Machine Learning, Mathematics, etc. Indeed, the theoretical and practical advances in SAT research over the past twenty years have contributed to making SAT technology an indispensable tool in a variety of domains.

SAT 2023 welcomed scientific contributions addressing different aspects of SAT, including (but not restricted to) theoretical advances (such as exact algorithms, proof complexity, and other complexity issues), practical search algorithms, knowledge compilation, implementation-level details of SAT solvers and SAT-based systems, problem encodings and reformulations, applications (including both novel application domains and improvements to existing approaches), as well as case studies and reports on findings based on rigorous experimentation.

A total of 67 papers, comprising 47 regular papers, 10 short papers, and 10 tool papers, were submitted to and reviewed for SAT 2023. Each submission was reviewed by (at least) three Program Committee members and their selected external reviewers. The review process included an author response period, during which the authors of submitted papers were given the opportunity to respond to the initial reviews of their submissions. To reach a final decision, a Program Committee discussion period followed the author response period. External reviewers supporting the Program Committee were also invited to participate directly in the discussions for the papers they reviewed. Finally, 30 papers were accepted, of which 21 are regular papers, 3 are short papers, and 6 are tool papers.

In addition to the presentations of the accepted papers, the scientific program of SAT included two invited talks from Albert Atserias (Universitat Politecnica de Catalunya, Barcelona, Spain) and Ryan Williams (Massachusetts Institute of Technology, USA).

The conference hosted various associated events. In particular, the following three workshops and one tutorial, affiliated with SAT 2023, were held on July 4, 2023:

- Pragmatics of SAT (PoS 2023), organized by Matti Järvisalo and Daniel Le Berre.
- The International Workshop on Counting and Sampling (MCW 2023), organized by Johannes K. Fichte, Markus Hecher, and Kuldeep Meel.
- The International Workshop on Quantified Boolean Formulas and Beyond (QBF 2023), organized by Hubie Chen, Luca Pulina, Martina Seidl, and Friedrich Slivovsky.
- A Tutorial on OptiLog, by Carlos Ansótegui.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The results of several competitive events were also announced at SAT 2023:

- The Model Counting Competition 2023 (MC 2023), organized by Markus Hecher and Johannes K. Fichte.
- The QBF Gallery 2023, organized by Luca Pulina and Martina Seidl.
- The MaxSAT Evaluation 2023, organized by Matti Järvisalo, Jeremias Berg, Ruben Martins, and Andreas Niskanen.
- The SAT Competition 2023, organized by Marijn Heule, Markus Iser, Matti Järvisalo, Martin Suda, and Tomáš Balyo.

We thank everyone who contributed to making SAT 2023 a success. In particular, we thank the Local Arrangements Committee members Luca Pulina and Laura Pandolfo; the web manager Dario Guidotti; and all the organizers of the SAT affiliated workshops and competitions. We thank the invited speakers for readily accepting our invitation and delivering insightful talks.

We are indebted to the Program Committee members and the external reviewers, who dedicated their time to review and evaluate the submissions to the conference. We thank the authors of all submitted papers for their contributions, the SAT Association for their guidance and support in organizing the conference, the EasyChair conference management system for facilitating the submission and selection of papers as well as scheduling the final program, and the staff at LIPIcs for coordinating and assisting with the assembly of these proceedings.

We sincerely thank the sponsors of SAT 2023: The Artificial Intelligence journal for providing travel support to students attending the conference, the University of Sassari for financial and organizational support, and Filuta AI and Amazon Web Services for their financial support.

July 2023 Meena Mahajan and Friedrich Slivovsky

Awards

At SAT 2023, several outstanding contributions and individuals within the community were acknowledged by the SAT 2023 Program Committee and the SAT Association.

- Despite the high quality of work submitted this year, no single paper distinctly stood out to warrant a **Best Paper Award**. However, three papers were highlighted for having received particular attention from the Program Committee for their noteworthy contributions. The **Highlighted Papers** are the following:

  - "Polynomial Calculus for MaxSAT",
    by Ilario Bonacina, María-Luisa Bonet, and Jordi Levy;
  - "Certified Knowledge Compilation with Application to Verified Model Counting",
    by Randal Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn Heule;
  - "IPASIR-UP: User Propagators for CDCL",
    by Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere.

  We commend the authors of these papers for their valuable work.

- The **Best Student Paper Award** was presented to Benjamin Böhm for the paper titled "QCDCL vs QBF Resolution: Further Insights".

- The **Test-of-Time Award** is given by the SAT Association for the most influential paper published in the SAT Conference 20 ± 1 years ago. This year, the award was presented to Armin Biere for the paper "Resolve and Expand", presented at SAT 2004.

- The SAT Association conferred a **Distinguished Service Award** on John Franco in honour of his long-lasting and foundational contributions to the series of International Conferences on Theory and Applications of Satisfiability Testing (SAT), the SAT association, and the Journal on Satisfiability, Boolean Modeling and Computation (JSAT).

Program Committee Members

Jeremias Berg	University of Helsinki
Olaf Beyersdorff	Friedrich Schiller University Jena
Armin Biere	University of Freiburg
Nikolaj Bjørner	Microsoft Research
Shaowei Cai	Chinese Academy of Sciences
Leroy Chew	TU Wien
Johannes Fichte	TU Wien
Marijn Heule	Carnegie Mellon University
Alexey Ignatiev	Monash University
Markus Iser	University of Helsinki
Mikoláš Janota	TU Prague
Jie-Hong Roland Jiang	National Taiwan University
Matti Järvisalo	University of Helsinki
Daniela Kaufmann	TU Wien
Benjamin Kiesl-Reiter	Amazon Web Services
Oliver Kullmann	Swansea University
Daniel Le Berre	CRIL-CNRS Université d'Artois
Inês Lynce	INESC-ID/IST
Meena Mahajan	The Institute of Mathematical Sciences
Felip Manyà	IIIA-CSIC
Ruben Martins	Carnegie Mellon University
Kuldeep Meel	National University of Singapore
Stefan Mengel	CNRS, CRIL, Lens
Alexander Nadel	Intel and Technion
Aina Niemetz	Stanford University
Jakob Nordström	University of Copenhagen and Lund University
Tomáš Peitl	TU Wien
Luca Pulina	University of Sassari
Christoph Scholl	University of Freiburg
Martina Seidl	Johannes Kepler University Linz
Laurent Simon	Bordeaux Institute of Technology
Friedrich Slivovsky	TU Wien
Mate Soos	Ethereum Foundation
Martin Suda	Czech Technical University in Prague
Stefan Szeider	TU Wien
Marc Vinyals	University of Auckland

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Adrian Rebola-Pardo
Agnes Schleitzer
Alex Ozdemir
Alexis de Colnet
Amalee Wilson
Andre Schidler
Andreas Niskanen
Andy Oertel
Antonio Morgado
Arijit Shaw
Arne Meier
Benjamin Böhm
Bernhard Gstrein
Carlos Mencía
Che Cheng
Chia-Hsuan Su
Christoph Berkholz
Christoph Jabs
Chun-Yu Wei
Dario Guidotti
Emre Yolcu
Filipe Gouveia
Florent Capelli
Franz Reichl
Haoze Wu
Jan Hůla
Jiong Yang
Jonas Conneryd
Jordi Coll
Kaspar Kasche
Kenji Hashimoto
Kuo-Wei Ho
Luc Spachmann
Margarida Ferreira
Massimo Lauria
Mathias Fleury
Mohamed Sami Cherif
Mohimenul Kabir
Rafael Kiesel
Shuo Pang
Sibylle Möhle
Steffen Meier
Supratik Chakraborty
Thibault Gauthier
Tim Hoffmann
Timothy van Bremen
Tobias Philipp
Tobias Seufert
Tomas Balyo
Valentin Roland
Yu-Wei Fan
Yun-Rong Luo

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Markus Anders (1)
TU Darmstadt, Germany

Jeremy Avigad (6)
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA

Teodora Baluta (28)
National University of Singapore, Singapore

Olaf Beyersdorff (2, 4)
Friedrich-Schiller-Universität Jena, Germany

Armin Biere (3, 8, 21)
Universität Freiburg, Germany

Ilario Bonacina (5)
Polytechnic University of Catalonia, Barcelona, Spain

Maria Luisa Bonet (5)
Polytechnic University of Catalonia, Barcelona, Spain

Randal E. Bryant (6)
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Matthew Burns (25)
Department of Electrical and Computer Engineering, University of Rochester, NY, USA

Benjamin Böhm (4)
Friedrich-Schiller-Universität Jena, Germany

Jonathan Chung (18)
University of Waterloo, Canada

Alexis de Colnet (7)
Algorithms and Complexity Group, TU Wien, Austria

Katalin Fazekas (8)
TU Wien, Austria

Noah Fleming (27)
Memorial University of Newfoundland, St. John's, Canada

Mathias Fleury (21)
Universität Freiburg, Germany

Dror Fried (9)
Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Nils Froleyks (3)
Johannes Kepler Universität, Linz, Austria

Long-Hin Fung (10)
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Vijay Ganesh (18, 27)
University of Waterloo, Canada

Harrison Green (11)
Carnegie Mellon University, Pittsburgh, PA, USA

Andrew Haberlandt (11)
Carnegie Mellon University, Pittsburgh, PA, USA

Bo He (29)
School of Information Science and Technology, Northeast Normal University, Changchun, China

Maximilian Heisinger (24)
Johannes Kepler Universität Linz, Austria

Marijn J. H. Heule (6, 11)
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Tim Hoffmann (2)
Friedrich-Schiller-Universität Jena, Germany

Michael C. Huang (25)
Department of Electrical and Computer Engineering, University of Rochester, NY, USA

George Katsirelos (12)
Université Paris-Saclay, AgroParisTech, INRAE, UMR MIA Paris-Saclay, 91120, Palaiseau, France

Markus Kirchweger (8, 13, 14)
TU Wien, Austria

Antonina Kolokolova (27)
Memorial University of Newfoundland, St. John's, Canada

Jordi Levy (5)
Artificial Intelligence Research Institute, Spanish Research Council (IIIA-CSIC), Barcelona, Spain

Chunxiao Li (18, 27)
University of Waterloo, Canada

Jiaxin Liang (29)
School of Information Science and Technology, Northeast Normal University, Changchun, China

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Juraj Major (23)
Masaryk University, Brno, Czech Republic

Vasco Manquinho (19)
INESC-ID, Instituto Superior Técnico,
University of Lisbon, Portugal

Ruben Martins (19)
Carnegie Mellon University,
Pittsburgh, PA, USA

Gabriele Masina (15)
DISI, University of Trento, Italy

Kuldeep S. Meel (28)
National University of Singapore, Singapore

Stefan Mengel (16)
Univ. Artois, CNRS, Centre de Recherche en
Informatique de Lens (CRIL), France

Alexander Nadel (9, 17)
Intel Corporation, Haifa, Israel;
Faculty of Data and Decision Sciences,
Technion, Haifa, Israel

Wojciech Nawrocki (6)
Department of Philosophy, Carnegie Mellon
University, Pittsburgh, PA, USA

Aina Niemetz (8)
Stanford University, CA, USA

Albert Oliveras (18)
Technical University of Catalonia,
Barcelona, Spain

Pedro Orvalho (19)
INESC-ID, Instituto Superior Técnico,
University of Lisbon, Portugal

Tomáš Peitl (13)
Algorithms and Complexity Group,
TU Wien, Austria

Andreas Plank (20)
Johannes Kepler Universität Linz, Austria

Florian Pollitt (21)
Universität Freiburg, Germany

Mathias Preiner (8)
Stanford University, CA, USA

Adrián Rebola-Pardo (22)
Technische Universität Wien, Austria;
Johannes Kepler Universität Linz, Austria

Manfred Scheucher (14)
Institut für Mathematik, Technische Universität
Berlin, Germany

Tereza Schwarzová (23)
Masaryk University, Brno, Czech Republic

Pascal Schweitzer (1)
TU Darmstadt, Germany

Roberto Sebastiani (15)
DISI, University of Trento, Italy

Martina Seidl (20, 24)
Johannes Kepler Universität Linz, Austria

Irfansha Shaik (24)
Aarhus University, Denmark

Yogev Shalmon (9)
Intel Corporation, Haifa, Israel;
The Open University of Israel, Ra'anana, Israel

Anshujit Sharma (25)
Department of Electrical and Computer
Engineering, University of Rochester, NY, USA

Arijit Shaw (28)
Chennai Mathematical Institute, India;
IAI, TCG-CREST, Kolkata, India

Mate Soos (1, 28)
National University of Singapore, Singapore

Luc Nicolas Spachmann (2)
Friedrich-Schiller-Universität Jena, Germany

Giuseppe Spallitta (15)
DISI, University of Trento, Italy

Jan Strejček (23)
Masaryk University, Brno, Czech Republic

Stefan Szeider (8, 13, 14)
TU Wien, Austria

Tony Tan (10)
Department of Computer Science and
Information Engineering, National Taiwan
University, Taipei, Taiwan

Jacobo Torán (26)
Institut für Theoretische Informatik,
Universität Ulm, Germany

Jaco van de Pol (24)
Aarhus University, Denmark

Marc Vinyals (27)
University of Auckland, New Zealand

Ruiwei Wang (30)
National University of Singapore, Singapore

Wenxi Wang (3)
University of Texas at Austin, TX, USA

Darryl Wu (18)
University of Waterloo, Canada

Florian Wörz (26)
Institut für Theoretische Informatik,
Universität Ulm, Germany

Jiong Yang (28)
National University of Singapore, Singapore

Roland H. C. Yap (30)
National University of Singapore, Singapore

Minghao Yin (29)
School of Information Science and Technology,
Northeast Normal University, Changchun,
China;
Key Laboratory of Applied Statistics of MOE,
Northeast Normal University, Changchun, China

Junping Zhou (29)
School of Information Science and Technology,
Northeast Normal University, Changchun, China

Neng-Fa Zhou (30)
The City University of New York, NY, USA;
Relational AI, Berkeley, CA, USA

Algorithms Transcending the SAT-Symmetry Interface

Markus Anders
TU Darmstadt, Germany

Pascal Schweitzer
TU Darmstadt, Germany

Mate Soos
National University of Singapore, Singapore

Abstract

Dedicated treatment of symmetries in satisfiability problems (SAT) is indispensable for solving various classes of instances arising in practice. However, the exploitation of symmetries usually takes a black box approach. Typically, off-the-shelf external, general-purpose symmetry detection tools are invoked to compute symmetry groups of a formula. The groups thus generated are a set of permutations passed to a separate tool to perform further analyzes to understand the structure of the groups. The result of this second computation is in turn used for tasks such as static symmetry breaking or dynamic pruning of the search space. Within this pipeline of tools, the detection and analysis of symmetries typically incurs the majority of the time overhead for symmetry exploitation.

In this paper we advocate for a more holistic view of what we call the *SAT-symmetry interface*. We formulate a computational setting, centered around a new concept of joint graph/group pairs, to analyze and improve the detection and analysis of symmetries. Using our methods, no information is lost performing computational tasks lying on the SAT-symmetry interface. Having access to the entire input allows for simpler, yet efficient algorithms.

Specifically, we devise algorithms and heuristics for computing finest direct disjoint decompositions, finding equivalent orbits, and finding natural symmetric group actions. Our algorithms run in what we call instance-quasi-linear time, i.e., almost linear time in terms of the input size of the original formula and the description length of the symmetry group returned by symmetry detection tools. Our algorithms improve over both heuristics used in state-of-the-art symmetry exploitation tools, as well as theoretical general-purpose algorithms.

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

Keywords and phrases boolean satisfiability, symmetry exploitation, computational group theory

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.1

Related Version *Full Version*: `https://arxiv.org/abs/2306.00613` [5]

Funding The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148).

1 Introduction

Many SAT instances, especially of the hard combinatorial type, exhibit symmetries. When symmetries exhibited by these instances are not handled adequately, SAT solvers may repeatedly explore symmetric parts of the search space. This can dramatically increase runtime, sometimes making it impossible for the solver to finish within reasonable time [7].

One common method to handle the symmetries is to add symmetry breaking formulas to the problem specification [10, 2]. This approach is called static symmetry breaking. Another, competing, approach is to handle symmetries dynamically during the running of the SAT solver. There are a variety of such dynamic strategies, exploiting symmetry information

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 1; pp. 1:1–1:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

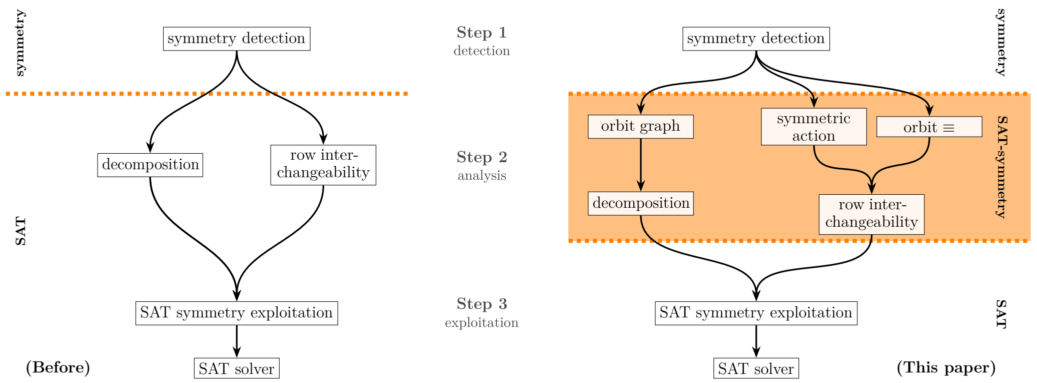


Figure 1 Blurring the lines of the SAT-symmetry interface. We analyze existing practical routines (left), draw connections to existing concepts in computational group theory, and describe improved algorithms in our new SAT-symmetry context (right).

during variable branching [21] and learning [28, 12]. For SAT, the tools Shatter [1] and BreakID [13, 8] take the static symmetry breaking approach, while SymChaff [28] and SMS [21] take the dynamic symmetry exploitation approach.

While there is a growing number of competing approaches of how best to handle symmetries, there are also a number of common obstacles: symmetries of the underlying formula have to be detected first, and the structure of symmetries has to be understood, at least to some degree. Approaches that handle symmetries can be typically divided into three distinct steps: (Step 1) symmetry detection, (Step 2) symmetry analysis, and (Step 3) symmetry breaking, or other ways of exploiting symmetry. In the following, we discuss each of these steps, also illustrated on the left side of Figure 1.

Step 1. In practice, symmetries are detected by modeling a given SAT formula as a graph, and then applying an off-the-shelf symmetry detection tool, such as saucy [11], to the resulting graph. Since symmetries form a permutation group under composition, a symmetry detection tool does not return all the symmetries. Instead, it only returns a small set of *generators*, which, by composition, lead to all the symmetries of the formula. Indeed, returning only a small set of generators is crucial for efficiency, since the number of symmetries is often exponential in the size of the formula.

Step 2. Symmetry exploitation algorithms apply heuristics to analyze the structure of the group described by the generators. This is necessary to enable the best possible use of the symmetries to improve SAT solver performance. We mention three examples for structural analyzes. Firstly, the *disjoint direct decomposition* splits a group into independent parts that can be handled separately. Secondly, so-called *row interchangeability* subgroups of the group [13, 14, 25] are of particular interest since they form a class of groups for which linear-sized, complete symmetry breaking constraints are known. Thirdly, *stabilizers* are commonly used for various purposes among both static and dynamic approaches [27].

Step 3. Lastly, the symmetries and structural insights are used to reduce the search space in SAT using one of the various static and dynamic symmetry exploitation approaches.

Designing symmetry exploitation algorithms typically involves delicately balancing computational overhead versus how thoroughly symmetries are used. In this trade-off, symmetry detection (Step 1) and analysis (Step 2) typically induce the majority of the overhead [13]. The main focus of this paper is improving the analysis of symmetries, i.e. (Step 2).

Practical implementations in use today that perform such structural analyzes do so through heuristics. While using heuristics is not an issue per se, some heuristics currently in use strongly depend on properties that the generators returned by symmetry detection tools may or may not exhibit. For example, BreakID and the MIP heuristic in [25] both rely on so-called *separability* of the generating set and a specific arrangement of *transpositions* being present. Neither of these properties are guaranteed by contemporary symmetry detection tools [9].

In fact, modern symmetry detection tools such as Traces [23] and dejavu [3] return randomly selected symmetries, since the use of randomization provides an asymptotic advantage in the symmetry detection process itself [4]. However, generating sets consisting of randomly selected symmetries are in a sense the exact opposite of what is desired for the heuristics, since with high probability random symmetries satisfy neither of the required conditions. This is particularly unfortunate, as dejavu is currently the fastest symmetry detection tool available for graphs stemming from SAT instances [6].

Another downside of the use of practical heuristics for the structural analysis of the group is that they are often also computationally expensive and make up a large portion of the runtime of the overall symmetry exploitation process. For example, the row interchangeability algorithm of BreakID performs multiple callbacks to the underlying symmetry detection tool, where each call can be expensive.

Altogether, heuristics in use today sometimes cause significant overhead, while also posing an obstacle to speeding up symmetry detection itself. This immediately poses the question: why is it that these heuristics are currently in place that cause such a loss of efficiency when it comes to computations within the SAT-symmetry interface?

We believe that the issue is that tools on either side of the interface treat each other as a black box. Indeed, when considered as an isolated task, algorithms for the analysis of permutation groups are well-researched in the area of computational group theory [30]. Not only is the theory well-understood, but there are also highly efficient implementations [15]. However, we can make two crucial observations regarding the available algorithms. First and foremost, for group theoretic algorithms from the literature that are deemed to have linear or nearly-linear runtime [30], the concrete runtime notions actually differ from the ones applicable in the overall context. In fact, the runtime is essentially measured in terms of a dense rather than a sparse input description. Therefore, in the context of SAT-solving or graph algorithms, the runtime of these algorithms should rather be considered quadratic. Secondly, in computational group theory, algorithms assume that only generators for an input group are available. However, in the context of the SAT-symmetry interface, not only a group but also a graph (computed from the original formula) is available. It turns out as a key insight of our paper that lacking access to the graphs crucially limits the design space for efficient algorithms.

Contributions. Advocating a holistic view of the SAT-symmetry interface, we develop algorithms that transcend both into the SAT domain and the symmetry domain at the same time. This is illustrated in Figure 1 on the right side.

Firstly, we provide a definition for the computational setting such as input, output, and runtime, under which these algorithms should operate (Section 3). We then extract precise formal problem definitions from heuristics implemented in state-of-the-art tools (Section 4). Lastly, we demonstrate the efficacy of our new approach by providing faster theoretical algorithms for commonly used heuristics, as is described below.

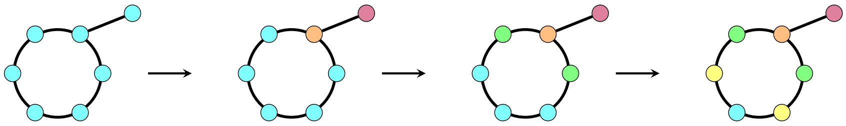


Figure 2 An illustration of a color refinement process.

Computational Setting. In our new computational setting, algorithms take as input a *joint graph/group pair*, meaning a group S and corresponding graph G, whose symmetry group is precisely $\langle S \rangle$. We define a precise notion of *instance-linear time*, meaning it is linear in the encoding size of the SAT formula, graph, and group.

New Algorithms. Given a joint graph/group pair, we develop and analyze the following algorithms:

A1 An instance-linear algorithm for computing the *finest direct disjoint decomposition* of the symmetry group of a graph (Section 5). We also give a heuristic specific to SAT formulas, decomposing the symmetry group on the literals.

A2 An algorithm to simultaneously detect *natural symmetric group actions* on all the orbits of a group (Section 6). Here we exploit randomized techniques from computational group theory for the detection of "giant" permutation groups. We give instance-linear heuristics which are able to exploit properties of the SAT-symmetry interface.

A3 An instance-quasi-linear algorithm to compute *equivalent symmetric orbits*, under some mild assumptions about the generating set (Section 7). In conjunction with (A2), this enables us to detect *all* elementary row interchangeability subgroups.

Both (A1) and (A3) improve the (at least) quadratic runtime of previous, general-purpose permutation group algorithms of [9] and [30], respectively.

2 Preliminaries and Related Work

Graphs and Symmetries. A colored graph $G = (V, E, \pi)$ consists of a set of vertices V, edges $E \subseteq V \times V$, and a vertex coloring $\pi \colon V \to C$ which maps V to some set of colors C. We use $V(G)$, $E(G)$, and $\pi(G)$ to refer to the vertices, edges, and coloring of G, respectively.

A symmetry, or *automorphism*, of a colored graph $G = (V, E, \pi)$ is a bijection $\varphi \colon V \to V$ such that $\varphi(E) = E$ as well as $\pi(v) = \pi(\varphi(v))$ for all $v \in V$. In other words, symmetries preserve the neighborhood relation of the graph, as well as the coloring of vertices. The colors of vertices in the graph are solely used to ensure that distinctly colored vertices are not mapped onto each other using symmetries. Together, all symmetries of a graph form a permutation group under composition, which we call the *automorphism group* $\mathrm{Aut}(G)$.

In this paper, we call software tools computing the automorphism group of a graph *symmetry detection tools* [23, 11, 18, 23, 3]. In the literature, these tools are also often called *practical graph isomorphism solvers*. In this paper, we avoid the use of this term in order not to confuse them with SAT *solvers*.

Color refinement. A common algorithm applied when computing the symmetries of a graph is *color refinement*. Given a colored graph $G = (V, E, \pi)$, color refinement *refines* the coloring π of G into $G' = (V, E, \pi')$. Crucially, the automorphism group remains invariant under color refinement, i.e., $\mathrm{Aut}(G) = \mathrm{Aut}(G')$.

We now describe the algorithm. If two vertices in some color $X = \pi^{-1}(c)$ have a different number of neighbors in another color $Y = \pi^{-1}(c')$, then X can be split by partitioning it according to neighbor counts in Y. After the split, two vertices have the same color precisely if they had the same color before the split, and they have the same number of neighbors in Y.

We repeatedly split classes with respect to other classes until no further splits are possible. Figure 2 shows an illustration of the color refinement procedure. A coloring which does not admit further splits is called *equitable*. For a graph G, color refinement can be computed in time $\mathcal{O}(|E(G)| \log |V(G)|)$ [22, 26].

Let us also recall a different definition for equitable colorings: A coloring π of a graph is equitable if for all pairs of (not necessarily distinct) color classes C_1, C_2, all vertices in C_1 have the same number of neighbors in C_2 (i.e., $|N(v) \cap C_2| = |N(v') \cap C_2|$ for all $v, v' \in C_1$). Given a coloring π, color refinement computes an equitable refinement π', i.e., an equitable coloring π' for which $\pi'(v) = \pi'(v')$ implies $\pi(v) = \pi(v')$. In fact, it computes the *coarsest* equitable refinement.

Permutation Groups. The *symmetric group* $\mathrm{Sym}(\Omega)$ is the permutation group consisting of all permutations of the set Ω. A *permutation group* on *domain* Ω is a group Γ that is a subgroup of $\mathrm{Sym}(\Omega)$, denoted $\Gamma \leq \mathrm{Sym}(\Omega)$. For a subset of the domain $\Omega' \subseteq \Omega$, the *restriction* of Γ to Ω' is $\Gamma|_{\Omega'} \coloneqq \{\varphi|_{\Omega'} \mid \varphi \in \Gamma\}$ (where $\varphi|_{\Omega}$ denotes restricting the domain of φ to Ω). The restriction is not necessarily a group since the images need not be in Ω'. The *pointwise stabilizer* is the group $\Gamma_{(\Omega')} \coloneqq \{\varphi \in \Gamma \mid \forall p \in \Omega' : \varphi(p) = p\}$, obtained by fixing all points of Ω' individually.

Whenever we are dealing with groups, we use a specific, succinct encoding. Instead of explicitly representing each element of the group, we only store a subset that is sufficient to obtain any other element through composition. Formally, let S be a subset of the group Γ, i.e., $S \subseteq \Gamma$. We call S a *generating set* of Γ whenever we obtain precisely Γ when exhaustively composing elements of S. We write $\langle S \rangle = \Gamma$. Moreover, each individual element $\varphi \in S$ can be referred to as a *generator* of Γ.

We write $\mathrm{supp}(\varphi) \coloneqq \{\omega \mid \omega \in \Omega \wedge \varphi(\omega) \neq \omega\}$ for the *support of a map*, meaning points of Ω not fixed by φ. The *support of a group* $\Gamma \in \mathrm{Sym}(\Omega)$ is the union of all supports of elements of Γ, i.e., $\mathrm{supp}(\Gamma) \coloneqq \{\omega \mid \omega \in \Omega \wedge \exists \varphi \in \Gamma : \varphi(\omega) \neq \omega\}$.

We use the *cycle notation* for permutations $\varphi \colon \Omega \to \Omega$. The permutation of $\{1, \dots, 5\}$ given by $1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 1, 4 \mapsto 5, 5 \mapsto 4$ we write as $(1,2,3)(4,5)$. Note that, for example $(1,2,3)(5,4)$ and $(3,1,2)(4,5)$ denote the same permutation. Algorithmically the cycle notation enables us to read and store a permutation φ in time $\mathrm{supp}(\varphi)$.

When considering two permutation groups Γ and Γ' it is possible that the groups are isomorphic as abstract groups but not as permutation groups. For example, if we let the symmetric group $\mathrm{Sym}(\Omega)$ act component-wise on pairs of elements of Ω, we obtain a permutation group with domain Ω^2 that also has $|\Omega|!$ many elements. In fact this group is isomorphic to $\mathrm{Sym}(\Omega)$ as an abstract group. We say a group Γ is a *symmetric group in natural action* if the group is $\mathrm{Sym}(\Omega)$, where Ω is the domain of Γ.

SAT and Symmetries. A Boolean satisfiability (SAT) instance F is commonly given in *conjunctive normal form* (CNF), which we denote with $F = \{(l_{1,1} \vee \dots \vee l_{1,k_1}), \dots, (l_{m,1} \vee \dots \vee l_{m,k_m})\}$, where each element of F is called a clause. A clause itself consists of a set of *literals*. A literal is either a variable or its negation. We use $\mathrm{Var}(F) \coloneqq \{v_1, \dots, v_n\}$ for the set of *variables* of F and we use $\mathrm{Lit}(F)$ for its literals.

A symmetry, or *automorphism*, of F is a permutation of the literals $\varphi \colon \mathrm{Lit}(F) \to \mathrm{Lit}(F)$ satisfying the following two properties. First, it maps F back to itself, i.e., $\varphi(F) \equiv F$, where $\varphi(F)$ is applied element-wise to the literals in each clause. Here clauses are equivalent, if they are the same when treated as unordered sets of literals, for example $(x \vee y) \equiv (y \vee x)$. Then $F' \equiv F$ if F' is obtained from F by reordering the literals of F' within the clauses. Second, for all $l \in \mathrm{Lit}(F)$ it must hold that $\overline{\varphi(l)} = \varphi(\overline{l})$, i.e., φ induces a permutation of the

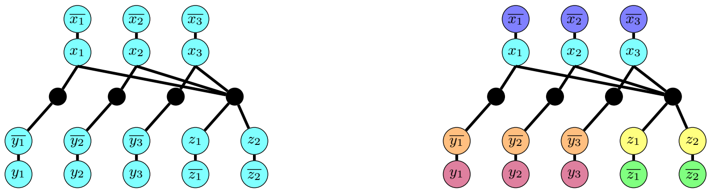

$\overline{x_1}$ $\overline{x_2}$ $\overline{x_3}$
x_1 x_2 x_3
$\overline{y_1}$ $\overline{y_2}$ $\overline{y_3}$ z_1 z_2
y_1 y_2 y_3 $\overline{z_1}$ $\overline{z_2}$
$\overline{x_1}$ $\overline{x_2}$ $\overline{x_3}$
x_1 x_2 x_3
$\overline{y_1}$ $\overline{y_2}$ $\overline{y_3}$ z_1 z_2
y_1 y_2 y_3 $\overline{z_1}$ $\overline{z_2}$

Figure 3 Example model graph $\mathcal{M}(F_E) = \mathcal{M}(\{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\})$. The automorphisms of the graph correspond to the automorphisms of the formula. The coloring on the right side shows the orbit partition.

variables. For example the permutation mapping x_i to $\neg x_{i+1}$ and x_i to x_{i+1}, with indices taken modulo 4, is a symmetry of $(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_4 \vee \neg x_1 \vee x_2 \vee \neg x_3)$. The permutation group of all symmetries of F is $\mathrm{Aut}(F) \leq \mathrm{Sym}(\mathrm{Lit}(F))$.

It is well understood that the symmetries of a SAT formula F can be captured by a graph. We call this the *model graph* and denote it with $\mathcal{M}(F)$. While there exists various constructions for the model graphs, we use the following common construction. Each literal $l \in \mathrm{Lit}(F)$ is associated with a vertex l. Each clause $C \in F$ is associated with a vertex C. All pairs of literals l and $\bar{l}$ are connected by an edge. For all literals $l \in C$ of a clause C, we connect vertices l and C. Lastly, to distinguish clause vertices from literal vertices, we color all clauses with color 0 and all literals with color 1. As desired, for this graph, $\mathrm{Aut}(F) = \mathrm{Aut}(\mathcal{M}(F))|_{\mathrm{Lit}(F)}$ holds [29].

Consider the formula $F_E \coloneqq \{(x_1 \vee \overline{y_1}), (x_2 \vee \overline{y_2}), (x_3 \vee \overline{y_3}), (x_1 \vee x_2 \vee x_3 \vee z_1 \vee z_2)\}$. Throughout the paper, we use F_E as our running example. Figure 3 shows its model graph. Regarding the symmetries of F_E, note that there are symmetries interchanging all of x_1, x_2, x_3, all of y_1, y_2, y_3 and of the z_1, z_2. A generating set S_E for $\mathrm{Aut}(F_E)$ is $S_E = \{(x_1, x_2, x_3)(\overline{x_1}, \overline{x_2}, \overline{x_3})(y_1, y_2, y_3)(\overline{y_1}, \overline{y_2}, \overline{y_3}), (x_1, x_2)(\overline{x_1}, \overline{x_2})(y_1, y_2)(\overline{y_1}, \overline{y_2}), (z_1, z_2)\}$.

Orbits. Given a permutation group $\Gamma \leq \mathrm{Sym}(\Omega)$, we denote with $\omega^\Gamma \subseteq \Omega$ the *orbit* of a point $\omega \in \Omega$. That is, an element $\omega' \in \Omega$ is in ω^Γ whenever there is a $\varphi \in \Gamma$ with $\varphi(\omega') = \omega$.

The orbits of our example $\mathrm{Aut}(F_E)$ are shown in Figure 3, e.g., the orbit $\{\overline{z_1}, \overline{z_2}\}$ is green.

3 SAT-Symmetry Computational Setting

Let us describe the computational setting in which our new algorithms operate. Since we want the theoretical runtimes to reflect more closely the runtimes in practice, there are two important differences compared to the traditional computational group theory setting. These differences are in the measure of runtime as well as in the format of the input.

Joint Graph/Group Pairs. Typically, algorithms in computational group theory dealing with permutation groups assume as their input a generating set of permutations S of a group $\Gamma = \langle S \rangle$. While this is certainly a natural setting when discussing algorithms for groups in general, in our setting this input format disregards further information that is readily available. Therefore, we require that algorithms in the SAT-symmetry interface have access to more information about the input group. Specifically, we may require that the input consists of *both* a generating set S and a graph G with $\langle S \rangle = \mathrm{Aut}(G)$. We call this a *joint graph/group pair* (G, S). For our SAT context, we may moreover assume that the SAT formula F with $\mathcal{M}(F) = G$ is available, whenever necessary.

Instance-Linear Runtime. In computational group theory, given a generating set S for a permutation group $\langle S\rangle \leq \mathrm{Sym}(\Omega)$, a runtime of $\mathcal{O}(|S||\Omega|)$ is typically considered linear time [30]. This is however only a very crude upper bound when seen in terms of the actual encoding size of a given generating set. In particular, when generators are sparse, as is common in SAT [29, 11], linear time in this sense is *not* necessarily linear time in the encoding size, which is what we would use in a graph-theoretic or SAT context.

Specifically, we are interested in measuring the runtime of algorithms relative to the encoding size of a generating set given in a sparse format. Therefore, we define the encoding size of a generating set S as $\mathrm{enc}(S) := \Sigma_{p\in S}|\,\mathrm{supp}(p)|$.

In particular, given a SAT formula F, graph $G = (V, E)$, and generating set S, the goal is to have algorithms that (ideally) run in time linear in $|F| + |V| + |E| + \mathrm{enc}(S)$. In order to not confuse the "types of linear time", we refer to such algorithms as *instance-linear*. Analogously, an algorithm has *instance-quasi-linear* time if it runs in time $\mathcal{O}((|F| + |V| + |E| + \mathrm{enc}(S)) \cdot (\log(|F| + |V| + |E| + \mathrm{enc}(S)))^c)$ for some constant c.

Illustrative Examples. The task of computing the orbits is an excellent example demonstrating the usefulness instance-quasi-linear time. As transitive closure, we can find the orbit Δ of an element in time $\mathcal{O}(|\Delta||S|)$ [30]. However, with instance-quasi-linear time in mind, we quickly arrive at an algorithm to compute the entire orbit partition in time $\mathcal{O}(\mathrm{enc}(S) \cdot \alpha(\mathrm{enc}(S)))$ using a union-find data structure, where α is the inverse Ackermann function. The inverse Ackermann function exhibits substantially slower growth than $\log(n)$.

Furthermore, having access to the graph of a graph/group pair (G, S) gives a significant advantage in what is algorithmically possible. A good example of the difference is that testing membership $\varphi \in \langle S\rangle$ is much easier for the graph/group pair: testing $\varphi(G) = G$ (which is true if and only if $\varphi \in \langle S\rangle$) can be done in instance-linear time. However, testing $\varphi \in \langle S\rangle$ without access to the graph is much more involved. The best known method for the latter involves computing a strong generating set, corresponding base and Schreier table [30], followed by an application of the fundamental sifting algorithm [30]. Even performing only the last step of this process (sifting) is not guaranteed to be in instance-linear time.

4 Favorable Group Structures in SAT

We now propose problems which should be solved within the SAT-symmetry computational setting. We analyze heuristics used in advanced symmetry exploitation algorithms [13, 25, 17], extracting precise formal definitions.

Disjoint Direct Decomposition. Following [9], we say a direct product of a permutation group $\Gamma = \Gamma_1 \times \Gamma_2 \times \cdots \times \Gamma_r$ is a *disjoint direct decomposition* of Γ, whenever all Γ_i have pairwise disjoint supports. We call Γ_i a *factor* of the disjoint direct decomposition of Γ. A disjoint direct decomposition is *finest*, if we cannot decompose any factor further into a non-trivial disjoint direct decomposition. A recent algorithm solves the problem of computing the finest disjoint direct decomposition for permutation groups in polynomial-time [9].

In our running example, the finest disjoint direct decomposition of $\mathrm{Aut}(F_E)$ splits the group into a subgroup H_1 permuting only the x_i and y_i variables, and a subgroup H_2 permuting z_1 and z_2. Indeed, setting $H_1 = \langle\{(x_1, x_2, x_3)(\overline{x_1}, \overline{x_2}, \overline{x_3})(y_1, y_2, y_3)(\overline{y_1}, \overline{y_2}, \overline{y_3}), (x_1, x_2)(\overline{x_1}, \overline{x_2})(y_1, y_2)(\overline{y_1}, \overline{y_2})\}\rangle$, and $H_2 = \langle\{(z_1, z_2)\}\rangle$, we have $\mathrm{Aut}(F_E) = H_1 \times H_2$.

Computing a disjoint direct decomposition is a typical routine in symmetry exploitation tools [13, 25, 17]. It allows for separate treatment of each factor of the decomposition. The heuristics in use today do not guarantee that the decomposition is the finest disjoint direct decomposition: indeed, the heuristics of the tools mentioned above assume that the given generating sets are already *separable* [9]. This means it is assumed, that every generator $\varphi \in S$ only operates on one factor of the disjoint direct decomposition $\Gamma_1 \times \Gamma_2 \times \cdots \times \Gamma_r$. Formally, this means for each $\varphi \in S$ there is one $i \in \{1, \ldots, r\}$ for which $\mathrm{supp}(\varphi) \cap \mathrm{supp}(\Gamma_i) \neq \emptyset$ holds, and for all $j \in \{1, \ldots, r\}, j \neq i$ it holds that $\mathrm{supp}(\varphi) \cap \mathrm{supp}(\Gamma_i) = \emptyset$. For example, the generating set S_E we gave for $\mathrm{Aut}(F_E)$ is separable.

It is not known how often generating sets given for graphs of SAT formulas are separable for a given symmetry detection tool, or in particular after reducing the domain to the literals of the SAT formula. It is however obvious that for the most advanced general-purpose symmetry detection tools, Traces and dejavu, generators are not separable with very high probability due to the use of randomly selected generators [9].

Row Interchangeability. We now discuss the concept of row interchangeability [13, 14, 25]. Let F be a SAT formula. Let M be a variable matrix $M\colon \{1, \ldots, r\} \times \{1, \ldots, c\} \to \mathrm{Var}(F)$. We denote the entries of M with $x_{i,j}$ where $i \in \{1, \ldots, r\}$ and $j \in \{1, \ldots, c\}$. We define the shorthand $\mathrm{supp}(M) \coloneqq \{x_{i,j} \mid x_{i,j} \in M\} \cup \{\overline{x_{i,j}} \mid x_{i,j} \in M\}$. The set $\mathrm{supp}(M)$ denotes all the literals involved with the matrix M. We say F exhibits *row interchangeability* if there exists a matrix M such that for every permutation $\varphi \in \mathrm{Sym}(\{1, \ldots, r\})$, for the induced literal permutation $\varphi'\colon \mathrm{Lit}(F)|_{\mathrm{supp}(M)} \to \mathrm{Lit}(F)|_{\mathrm{supp}(M)}$ given by $x_{i,j} \mapsto x_{\varphi(i),j}, \neg x_{i,j} \mapsto \neg x_{\varphi(i),j}$ it holds that $\varphi' \in \mathrm{Aut}(F)|_{\mathrm{supp}(M)}$. Indeed, if this is the case, we can observe that the matrix M describes a subgroup of $\mathrm{Aut}(F)$ consisting of $\{\pi \in \mathrm{Aut}(F) \mid \exists \varphi \in \mathrm{Sym}(\{1, \ldots, r\}) : \varphi' = \pi|_{\mathrm{supp}(M)}\}$. We denote this group by $H_M \leq \mathrm{Aut}(F)$.

A crucial fact is that for $H_M|_{\mathrm{supp}(M)}$, linear-sized complete symmetry breaking is available [13, 14]. As is also in part discussed in [13, 14], we observe that the complete symmetry breaking for H_M is most effective whenever H_M is the only action on $\mathrm{supp}(M)$ in $\mathrm{Aut}(F)$, or more precisely, $\mathrm{Aut}(F)|_{\mathrm{supp}(M)} = H_M|_{\mathrm{supp}(M)}$. In this case, we call H_M an *elementary* row interchangeability subgroup. Otherwise, there are non-trivial symmetries $\varphi \in \mathrm{Aut}(F)|_{\mathrm{supp}(M)}$ with $\varphi \notin H_M|_{\mathrm{supp}(M)}$ or $\mathrm{supp}(M)$ is not a union of orbits. Indeed, in this case, the complete symmetry breaking of H_M might make it more difficult to break such overlapping symmetries φ: for example, if two row interchangeability subgroups H_M and $H_{M'}$ overlap, i.e., $\mathrm{supp}(M) \cap \mathrm{supp}(M') \neq \emptyset$, complete symmetry breaking can only be guaranteed for one of them using the technique of [13].

Whenever H_M is an elementary row interchangeability subgroup, the situation is much clearer: we can produce a linear-sized complete symmetry breaking formula and this covers at least all symmetries on the literals $\mathrm{supp}(M)$. In this paper, we therefore focus on computing elementary row interchangeability groups.

Let us consider F_E again: for the matrix $M \coloneqq \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}$ there is indeed a row interchangeability subgroup. (Recall that the group H_M permutes positive and negative literals of variables appearing in M.) For this example, H_M is both an elementary row interchangeability group and a factor in the finest direct disjoint decomposition of $\mathrm{Aut}(F_E)$.

Row Interchangeability and Equivalent Orbits. We now describe the matrix of elementary row interchangeability groups in more group-theoretic terms. We first define the notion of equivalent orbits:

▶ **Definition 1** (Equivalent orbits; see [30, Subsection 6.1.2]). *Two orbits Δ_1, Δ_2 are equivalent, if and only if there is a bijection $b\colon \Delta_1 \to \Delta_2$ such that for all $\varphi \in \Gamma$ and $\delta \in \Delta_1$, $\varphi(b(\delta)) = b(\varphi(\delta))$.*

We write $\Delta_1 \equiv \Delta_2$ to indicate orbits Δ_1 and Δ_2 are equivalent. It is easy to see this indeed defines an equivalence relation on the orbits [30].

We observe that if a row interchangeability subgroup H_M is elementary, each row of the matrix M is an orbit of $\mathrm{Aut}(F)$. Since all rows are moved simultaneously in the same way, we remark that rows of M are precisely equivalent orbits with a natural symmetric action:

▶ **Lemma 2.** *Let H_M be a row interchangeability subgroup of $\Gamma = \mathrm{Aut}(\mathcal{M}(F))$, and let $\Delta_i = (x_1, \ldots, x_c)$ denote a row of M. H_M is an elementary row interchangeability subgroup if and only if all of the following hold: (1) Δ_i is an orbit with a natural symmetric action in Γ. (2) For every other row Δ_j of M, it holds that $\Delta_i \equiv \Delta_j$. (3) For $\overline{\Delta_i} = (\overline{x_1}, \ldots, \overline{x_c})$, $\overline{\Delta_i}$ is also an orbit with $\overline{\Delta_i} \equiv \Delta_i$.*

There is an exact algorithm which computes equivalent orbits in essentially quadratic runtime [30]. Again, runtimes are difficult to compare due to different pre-conditions in [30]. In any case, the algorithm for equivalent orbits depends on computing a base and strong generating set, which is too slow from our perspective.

We may split detecting elementary row interchangeability groups into detecting *natural symmetric action* on the orbits, followed by computing *equivalent orbits*. We now turn to solving the problems defined above in the computational setting of the SAT-symmetry interface. Specifically, we propose algorithms for the finest disjoint direct decomposition (Section 5), natural symmetric action (Section 6), and equivalent orbits (Section 7).

5 Finest Disjoint Direct Decomposition

Having established the problems we want to address, we now turn to presenting suitable algorithms in the SAT-symmetry computational setting. In particular, recall that we want to make use of joint graph/group pairs in order to state algorithms that run in instance-quasi-linear time. We begin by computing the finest disjoint direct decomposition.

Specifically, given a joint graph/group pair (G, S), our aim is to compute the finest disjoint direct decomposition of the group $\langle S \rangle$. Our proposed algorithm, given the orbits, can do so in instance-linear time. The disjoint direct decomposition of a group allows us to separately treat each factor of the decomposition in symmetry exploitation or other consecutive algorithms.

To simplify the discussion, we assume the graph G to be undirected. However, the procedure generalizes to both directed and even edge-colored graphs.

Orbit Graph. We describe the *orbit graph*, which can be constructed from (G, S). We are particularly interested in the connected components of the orbit graph, which turn out to correspond exactly to the factors of the finest disjoint direct decomposition.

First, note that the orbit partition π of $\langle S \rangle$ can be viewed as a vertex coloring of the graph G, assigning to every vertex its orbit. We consider the graph $G' = (V(G), E(G), \pi)$, i.e., G colored with its orbit partition (see Figure 4, left).

We call two distinct orbits Δ, Δ' *homogeneously connected*, whenever either all vertices $v \in \Delta$ are adjacent to all vertices of Δ', or there is no edge with endpoints both in Δ and Δ'. Indeed, we could "flip edges" between homogeneously connected orbits such that they all become disconnected, without changing the automorphism group (see Figure 4, middle).

We now give the formal definition of the orbit graph. The orbit graph is essentially an adapted version of the so-called flipped quotient graph (see [20] for a discussion). The vertex set of the orbit graph is the set of orbits of $\langle S \rangle$, i.e., $\{\pi^{-1}(v) \mid v \in V(G')\}$. Two orbits Δ, Δ' are adjacent in the orbit graph if and only if the orbits are *not* homogeneously connected in the original graph G (see Figure 4, right).

Algorithm 1 Compute the orbit graph.

```
 1 function OrbitGraph($G'$)
       Input  : graph $G' = (V, E, \pi)$, where $\pi$ is the orbit partition of $\mathrm{Aut}(G')$
       Output : orbit graph $G_O$
 2     initialize integer array $A$ of size $|V|$ with all 0;
 3     initialize empty list $W$;
 4     initialize empty graph $G_O$;
 5     $V(G_O) := \pi(V)$;
 6     for ( $\Delta \in \pi(V)$ )
 7         pick an arbitrary $v \in \Delta$;
 8         for ( $(v, v') \in E$ )
 9             increment $A[\pi(v')]$;
10             add $\pi(v')$ to $W$;
11         for ( $\Delta' \in W$ )
12             if $A[\Delta'] > 0$ and $A[\Delta'] < |\pi^{-1}(\Delta')|$ then add edge $(\Delta, \Delta')$ in $G_O$ ;
13             $A[\Delta'] := 0$;
14     return $G_O$;
```

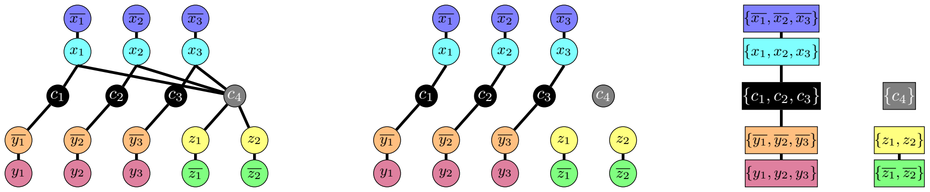


Figure 4 Model graph of F_E colored with its orbit partition on the left. The corresponding graph with flipped edges is in the middle, which disconnects parts of the graph. On the right the orbit graph is shown, whose 3 connected components correspond to the factors of the finest disjoint direct decomposition.

Description of Algorithm 1. Algorithm 1 describes how to compute the orbit graph from G'. The algorithm first initializes the graph G_O with a vertex set that contains exactly one vertex for each orbit of G'. It then counts for each orbit Δ, how many neighbors a vertex $v \in \Delta$ has in the other orbits. Since Δ is an orbit, this number is the same for all vertices, so it suffices to compute this for one $v \in \Delta$. Finally, the algorithm checks to which other colors the vertex v and thus the orbit Δ is *not* homogeneously connected (Line 12). If Δ and Δ' are not homogeneously connected, the edge (Δ, Δ') is added to the orbit graph.

Remark on the runtime of Algorithm 1. Using appropriate data structures for graphs (adjacency lists) and colorings (see [23], which in particular includes efficient ways to compute $|\pi^{-1}(C')|$), the algorithm can be implemented in instance-linear time.

Orbit Graph to Decomposition. Indeed, the connected components of the orbit graph represent precisely the factors of the finest disjoint direct decomposition of the automorphism group of the graph:

▶ **Lemma 3.** *Let* $\Gamma = \mathrm{Aut}(G)$*. The vertices represented by a connected component of the orbit graph of* G *are all in the same factor of the finest direct disjoint decomposition of* Γ *and vice versa.*

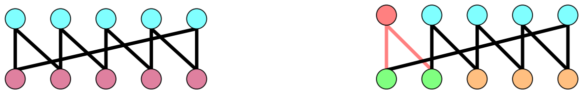


Figure 5 Illustration of two orbits that are non-homogeneously connected on the left in blue and purple. On the right, fixing a vertex of one orbit indicated in red immediately partitions the other orbit into two orbits: the neighbors of the red vertex in green, and the non-neighbors in orange.

Proof. Consider two orbits Δ_1, Δ_2 of Γ in different factors of any direct disjoint decomposition. Towards a contradiction, assume Δ_1, Δ_2 are not homogeneously connected. Note that naturally, the orbit coloring is equitable. Since the orbit coloring is equitable, the connection must be regular, i.e., each vertex of Δ_1 has d_1 neighbors in Δ_2, and every vertex of Δ_2 has d_2 neighbors in Δ_1 for some integers d_1, d_2. However, $0 < d_1 < |\Delta_2|$ and $0 < d_2 < |\Delta_1|$ hold.

Let us now fix a point $\delta \in \Delta_1$, i.e., consider the point stabilizer $\Gamma_{(\delta)}$. If two orbits are in different factors of a direct disjoint decomposition, fixing a point of Δ_1 must not change the group action on Δ_2. In particular, Δ_2 must be an orbit of $\Gamma_{(\delta)}$. However, δ is adjacent to some vertex $\delta' \in \Delta_2$ and non-adjacent to some vertex $\delta'' \in \Delta_2$ (see Figure 5 for an illustration). Having fixed δ, we can therefore not map d' to d''. This contradicts the assumption that Δ_1 and Δ_2 are in different factors of a direct disjoint decomposition. Hence, orbits in different factors of a direct disjoint decomposition must be homogeneously connected in G, i.e., non-adjacent in the orbit graph.

Now assume Δ_1 and Δ_k are in the same component in the orbit graph. Then, there must be a path of orbits $\Delta_1, \Delta_2, \dots, \Delta_k$ where each Δ_i, Δ_{i+1} is *not* homogeneously connected. In this case, we know for each $i \in \{1, \dots, k-1\}$ that Δ_i and Δ_{i+1} must be in the same factor of any disjoint direct decomposition. Therefore, Δ_1 and Δ_k must be in the same factor of every disjoint direct decomposition.

On the other hand, if Δ_1 and Δ_k are in different components in the orbit graph, they are in different factors of the finest disjoint direct decomposition. ◀

Since connected components can be computed in linear time in the size of a graph, and the size of the orbit graph is at most linear in the size of the original graph, we can therefore compute the finest direct disjoint decomposition in instance-linear time. In a consecutive step, the generators could be split according to factors, producing a separable generating set, again in instance-linear time. This is done by separating each generator into the different factors. Finally, given the finest direct disjoint decomposition, we can again produce a joint graph/group pair for each factor, by outputting for a factor H_i the induced subgraph $G'[H_i]$. We summarize the above in a theorem:

▶ **Theorem 4.** *Given a joint graph/group pair (G, S) and orbit partition of $\langle S \rangle$, there is an instance-linear algorithm which computes the following:*

1. *The finest disjoint direct decomposition $\langle S \rangle = H_1 \times H_2 \times \cdots \times H_m$.*
2. *A separable generating set S' with $\langle S' \rangle = G$.*
3. *For all factors $i \in \{1, \dots, m\}$ a joint graph/group pair (G_i, S_i) with $\mathrm{Aut}(G_i) = \langle S_i \rangle = H_i$ in instance-linear time.*

We recall that if the orbit partition of $\langle S \rangle$ is not yet available, we can compute it in instance-quasi-linear time.

Domain Reduction to SAT Literals. For a SAT formula F, we can apply the above procedure to its model graph $\mathcal{M}(F)$. However, as mentioned above, in SAT we are typically only interested in symmetries for a subset of vertices, namely the vertices that represent literals. Therefore, we are specifically interested in the finest direct disjoint decomposition of the automorphism group reduced to literal vertices $\mathrm{Aut}(\mathcal{M}(F))|_{\mathrm{Lit}(F)}$. The crucial point here is that when removing orbits that represent clauses, orbits of literal vertices can become independent and the disjoint direct decomposition can therefore become finer. We cannot simply apply our algorithm for the induced group $\mathrm{Aut}(\mathcal{M}(F))|_{\mathrm{Lit}(F)}$ since this is not a joint graph/group pair. Of course we could apply the algorithm from [9] that computes finest disjoint direct decomposition for permutation groups in general. However, we can detect *some* forms of independence by simple means using the original joint graph/group pair. Indeed, we will describe an algorithm that checks in instance-linear time whether the parts in a given partition of the literals are independent. We can thus at least check whether a given partition induces a disjoint direct decomposition (the proof can be found in the full version [5]):

▶ **Theorem 5.** *Let F be a CNF-Formula and $(\mathcal{M}(F), S)$ be a joint graph/group pair for the model graph of F. Given a partition $\mathrm{Lit}(F) = L_1 \cup L_2 \cup \cdots \cup L_t$ of the literals of F, the pair $(\mathcal{M}(F), S)$, and its orbits, we can check in instance-linear time whether the partition induces a disjoint direct product (that is, whether $\mathrm{Aut}(F) = \mathrm{Aut}(F)_{(L \setminus L_1)} \times \cdots \times \mathrm{Aut}(F)_{(L \setminus L_t)}$).*

6 Natural Symmetric Action

Before we can begin our discussion of the natural symmetric action, we need to discuss generating (nearly-)uniform random elements (see [30]) of a given permutation group $\langle S \rangle$. There is no known algorithm which produces uniform random elements of $\langle S \rangle$ in quasi linear time, even in computational group theory terminology [30]. However, there are multiple ways to produce random elements, most of which are proven to work well in practice, and can be implemented fairly easily [30, 15]. In this paper, we attempt to only make use of random elements sparingly. Whenever we do, as is common in computational group theory, we do not consider the particular method used to generate them and simply denote the runtime of the generation with μ. Moreover, we discuss potential synergies in the SAT-symmetry context which might help to avoid random elements in practice, whenever applicable.

We now explain how to efficiently test whether a permutation group is a symmetric group in natural action. Then, we describe more generally how to determine simultaneously for all orbits of a permutation group whether the induced action is symmetric in natural action.

Detecting symmetric permutation groups in their natural action is a well-researched problem in computational group theory. State-of-the-art practical implementations are available in modern computer algebra systems (such as in [24]). Typically, a natural symmetric action is detected using a so-called probabilistic *giant* test, followed by a test to ensure that the group is indeed symmetric. The tests work by computing (nearly) uniform random elements of the group and inspecting them for specific properties.

A permutation group is called a giant if it is the symmetric group or the alternating group in natural action. In many computational contexts, giants are by far the largest groups that appear, hence their name. Because of this, giants form bottleneck cases for various algorithms and therefore often need to be treated separately. To test whether a permutation group is a symmetric group in natural action, we first test whether the group is a giant.

We leverage the following facts:

▶ **Fact 6** (see [30, Corollary 10.2.2.]). *If a permutation group of degree n contains an element with a cycle of length p for some prime p with $n/2 < p < n - 2$ then G is a giant.*

▶ **Fact 7** (see [30, Corollary 10.2.3.]). *The proportion of elements in S_n containing a cycle of length p for a prime p with $n/2 < p < n-2$ is asymptotically* $1/\log(n)$.

Collectively, these statements show that we only need to generate few random elements of a group and inspect their cycle lengths to detect a giant. To then distinguish between the alternating group and the symmetric group, we can check whether all generators belong to the alternating group. This can be attained using basic routines, such as examining the so-called parity of a generator (see [30] for more details). Algorithm 2 generalizes the probabilistic test for a transitive group [30] to a test which is performed simultaneously to check for a natural symmetric action on all the orbits of a group.

Algorithm 2 Compute whether orbits induce a natural symmetric action.

```
function SymmetricAction($S, O$)
    Input  : generators $S$ with $\langle S \rangle = \Gamma \leq \mathrm{Sym}(\Omega)$, orbits $O \coloneqq \{\Delta_1, \dots, \Delta_m\}$ of $\Gamma$
    Output : set of orbits $S_O \subseteq O$, where $\Delta \in S_O$ induces a natural symmetric action
    // first, we filter orbits which can at most be alternating
    for ( $\Delta_i \in O$ )
        $t \coloneqq \top$;
        for ( $s \in S$ )
            if $s|_{\Delta_i} \notin \mathrm{Alt}(\Delta_i)$ then $t \coloneqq \bot$ ;          // $\Delta_i$ cannot be alternating
        if $t = \top$ then $O \coloneqq O \setminus \{\Delta_i\}$ ; // $\Delta_i$ cannot induce symmetric action
    // second, we test whether orbits are giants
    $S_O \coloneqq \emptyset$;
    for ( $\_ \in \{1, \dots, c\log(|\Omega|)^2\}$ )                      // repeat $c\log(|\Omega|)^2$ times
        $\varphi \coloneqq$ uniform random element of $\langle S|_{\cup_{\Delta_i \in S_O} \Delta_i} \rangle$;
        for ( $\Delta_i \in O$ )
            $p \coloneqq$ cycle length of longest cycle in $\varphi$ on $\Delta_i$;
            if $p > n/2 \land p < n-2 \land p$ prime then
                $S_O \coloneqq S_O \cup \{\Delta_i\}$;                    // $\Delta_i$ induces symmetric action
                $O \coloneqq O \setminus \{\Delta_i\}$;
    return $S_O$
```

Description of Algorithm 2. Overall, the algorithm samples uniform random elements of the group and checks whether the random elements exhibit long prime cycles (see Fact 6). More precisely, the algorithm first distinguishes between potential alternating and symmetric groups on each orbit. Then, it computes $d = c\log(|\Omega|)^2$ random elements. For each random element and each orbit, we then apply the giant test (Fact 6) to check whether the element certifies that the orbit induces a natural symmetric action.

Runtime of Algorithm 2. Let us assume access to random elements of the joint graph/group pair (G, S) with $\langle S \rangle \leq \mathrm{Sym}(\Omega)$ in time μ. Assuming a random element can be produced in time μ, the algorithm runs in worst-case time $\mathcal{O}(\log(|\Omega|)^2(\mu + |\Omega|))$.

Correctness of Algorithm 2. Regarding the correctness of the algorithm, the interesting aspect is to discuss the error probability. We argue that the error probability is at most $1/4$ if c is chosen to larger than $2\ln(2)$. Practical implementations use $c = 20$ in similar contexts [24].

If an orbit Δ does not induce a symmetric action, no error can be made. If an orbit Δ induces a symmetric action, by Fact 7, the probability that one iteration does not produce a long prime cycle for Δ is at most $(1 - 1/\log(n))$. Thus, the probability that none of the $c\log(|\Omega|)^2$ iterations produces a long prime cycle for Δ is bounded by $(1 - 1/\log(n))^{c\log(|\Omega|)^2} \leq (1/e)^{c\log(|\Omega|)} \leq 1/(4|\Omega|)$ since $c > 2\ln(2)$. Since there can be at most $|\Omega|$ orbits, using the union bound, we get that the probability that the test fails for at least one of the orbits is at most $|\Omega| \cdot 1/(4|\Omega|) = 1/4$.

When trying to compute a natural symmetric action on a graph/group pair, the following heuristics can be implemented in instance-linear time.

The first and most straightforward heuristic is that most of the time, it is fairly clear that the generators describe a natural symmetric action. In particular, symmetry detection based on depth-first search seem to often produce generators that are transpositions. From these symmetric actions can be detected immediately. This fact is implicitly used by column interchangeability heuristics in use today. There are however many more ways to detect a natural symmetric action, many of which are implemented in modern computer algebra systems such as [15, 24].

Next, the symmetry detection preprocessor SASSY [6] as well as the preprocessing used by TRACES sometimes detect a natural symmetric action on an orbit, by detecting certain structures of a graph. In these cases, the result should be immediately communicated to consecutive algorithms. We can also use the graph structure to immediately discard orbits from the test of Algorithm 2. In particular, all orbits Δ where $G[\Delta]$ is neither the empty graph nor the complete graph cannot have a natural symmetric action.

Furthermore, the generators produced by DEJAVU and TRACES are fairly random (for some parts even uniformly random [3]). This means they should presumably work well with the probabilistic tests above. Lastly, internally, symmetry detection tools often produce so-called Schreier-Sims tables [30], which can be used to produce random elements effectively.

Indeed, for our running example F_E, the natural symmetric action can be detected quite easily: let us consider the generators S_E reduced to the orbit of $\{x_1, x_2, x_3\}$. We observe that there is a generator (x_1, x_2, x_3) and (x_1, x_2). While this is not a set of generators detected by current symmetry exploitation algorithms [13, 25], this is indeed also an arguably obvious encoding of a natural symmetric action: for an orbit of size n, an n-cycle in conjunction with a transposition encodes a symmetric action.

7 Equivalent Orbits

Towards our overall goal to compute row interchangeability subgroups, we can now already determine which orbits induce a natural symmetric action. By Lemma 2, to detect elementary row interchangeability subgroups, we only miss a procedure for orbit equivalence.

We describe now how to compute equivalent orbits as the automorphism group of a special, purpose-built graph. Then, we give a faster algorithm computing equivalent orbits with natural symmetric action in instance-quasi-linear time, under mild assumptions. In particular, we can find all classes of equivalent orbits described by Lemma 2.

7.1 Cycle Type Graph

If two orbits are equivalent, they appear in every permutation in "the same manner": for example, if orbits Δ_1 and Δ_2 are equivalent then for every generator g, the cycle types g induces on Δ_1 are the same as the cycle types g induces on Δ_2. More generally, equivalent orbits must be equivalent with respect to *every* generating set of the group. We introduce the *cycle type graph* whose symmetries capture orbit equivalence. This means we can use a symmetry detection tool to detect equivalent orbits.

For a group $\Gamma \leq \mathrm{Sym}(\Omega)$ and generating set $\langle S \rangle = \Gamma$, we define the *cycle type graph* $\mathcal{C}(S)$ as follows.

Firstly, the **vertex set** of $\mathcal{C}(S)$ is the disjoint union $V(\mathcal{C}(S)) \coloneqq \Omega \dot\cup \dot{\bigcup}_{g \in S} \operatorname{supp}(g)$. In other words, there is a vertex for each element of Ω and there are separate elements for all the points moved by the generators. In particular if a point is moved by several generators there are several copies of the point.

Secondly, the **edges** of $\mathcal{C}(S)$ are added as follows: each corresponding vertex for $x \in \operatorname{supp}(g)$ is adjacent to the corresponding vertex for element $x \in \Omega$ via an undirected edge. Furthermore, there are directed edges $\{(x, g(x)) \mid x \in \operatorname{supp}(g)\}$. In other words, for each generator $s_i \in S$, we add directed cycles for each cycle of the generator, as shown in Figure 6a. In the following, we refer to directed cycles added in this manner as *cycle gadgets*.

Thirdly, we define a **vertex coloring** for $\mathcal{C}(S)$. For this we enumerate the generators, i.e., $S = \{s_1, \ldots, s_m\}$. We then color the vertices in Ω with color 0 and an $x \in \operatorname{supp}(g_i)$ is colored with (i, t), where t is the length of the cycle in g_i containing x. With this, the cycle type graph is constructed in such a way that its automorphism structure captures equivalence of orbits, as is described in more detail below.

We record several observations on automorphisms of the cycle type graph (missing proofs can be found in the full version [5]).

▶ **Lemma 8.** *If Δ_1, Δ_2 are orbits of Γ then there is some $b \in \operatorname{Aut}(\mathcal{C}(S))$ for which $b(\Delta_1) = \Delta_2$, if and only if $\Delta_1 \equiv \Delta_2$.*

We may formulate the observations in group theoretic terms, giving the following lemma.

▶ **Lemma 9.** *Given a group $\Gamma \leq S(\Omega)$, its centralizer in the symmetric group $C_{\operatorname{Sym}(\Omega)}(\Gamma)$ and the cycle type graph $\mathcal{C}(\Gamma)$ are a joint graph/group pair, i.e., $\operatorname{Aut}(\mathcal{C}(\Gamma)) = C_{S(\Omega)}(\Gamma)$.*

Since the centralizer of $\operatorname{Sym}(\Omega)$ in $\operatorname{Sym}(\Omega)$ is trivial for $|\Omega| > 2$, we get the following corollary.

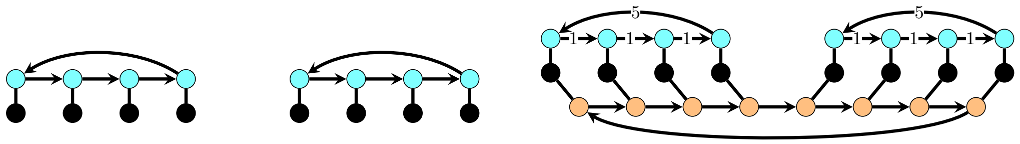
▶ **Corollary 10.** *If $\Gamma = \operatorname{Sym}(\Omega)$ with $|\Omega| > 2$, the cycle type graph of Γ is asymmetric.*

It follows from the corollary that for two equivalent orbits with a natural symmetric action the bijection b commuting with the generators and interchanging the orbits is in fact unique.

While the cycle type graph and the centralizer in the symmetric group $(\mathcal{C}(\Gamma), C_{\operatorname{Sym}(\Omega)}(\Gamma))$ is a joint graph/group pair, we still have to compute the group: so far, we only have access to $\mathcal{C}(\Gamma)$. One option is a symmetry detection tool. However, this goes against our goal of invoking symmetry detection tools unnecessarily often – and against our goal to find instance-linear algorithms. Hence, instead of computing the entire automorphism group, our approach is to make due with less: in the following, we enhance the cycle type graph in a way such that it becomes "easy" for color refinement. Color refinement is usually applied as a heuristic approximating the orbit partition of a graph. However, on the enhanced graphs, we prove that color refinement is guaranteed to compute the orbit partition. Then, we show that the orbit partition suffices to determine equivalent orbits. Overall, these methods are only guaranteed to work for orbits with a natural symmetric action, as is the case in elementary row interchangeability groups.

7.2 Symmetries of Cycle Type Graph with Unique Cycles

Our goal is now to enhance the cycle type graph such that color refinement is able to compute its orbit partition. This in turn enables us to detect equivalent orbits, and in turn elementary row interchangeability groups. Towards this goal, we first discuss an algorithm to compute



(a) Cycle type graph.

(b) Enhanced cycle type graph.

Figure 6 The cycle type graph and enhanced cycle type graph. The figure shows a cycle type gadget and canonical cyclic order for the permutation (1234)(5678). Automorphisms of this graph are elements of the centralizer, which indicate equivalent orbits.

unique cycles on orbits. Unique cycles are a key ingredient for our enhanced cycle type graph. These cycles should be invariant with respect to an ordered generating set, a concept we explain first.

Given an ordered generating set $(s_1, \dots, s_m)$ for a permutation group $\Gamma = \langle \{s_i \mid i \in \{1, \dots, m\}\} \rangle \leq \mathrm{Sym}(\Omega)$, a permutation $\varphi \in \mathrm{Sym}(\Omega)$ *fixes* the ordered generating set (pointwise under conjugation) if for all s_i we have $\varphi \circ s_i \circ \varphi^{-1} = s_i$.

A permutation $\pi \in \mathrm{Sym}(\Omega)$ is *invariant* with respect to the ordered generating set if all permutations φ that fix $(s_1, \dots, s_m)$ under conjugation also fix $(s_1, \dots, s_m, \pi)$ under conjugation[1]. Note that all group elements in $\pi \in \Gamma$ are invariant. However, there can be further invariant permutations.

We say a permutation π has a *unique cycle* if for some length $\ell > 1$, the permutation π contains exactly one cycle of ℓ. We now describe a two-step process. Step one is to compute an invariant permutation with a unique cycle. Step two is to use this to compute an invariant permutation with a cyclic order.

Unique Cycle from Generators. As a first step we now need an invariant *unique cycle* to proceed. We argue how to compute such a cycle for orbits on which our group induces a natural symmetric action.

We may use random elements to find a unique cycle. In fact, if we perform the giant test of Section 6, we get access to a unique cycle. However, in that section we needed a prime length cycle. If we are only interested in unique cycles, not necessarily of prime length, this process terminates much more quickly: *Golomb's constant* [16] measures, as $n \to \infty$, the probability that a random element $\varphi \in S_n$ has a cycle of length greater than $\frac{n}{2}$. The limit is greater than $\frac{4}{5}$.

In practice the existence of a unique cycle is a mild assumption: on the one hand some practical heuristics only apply if specific combinations of transpositions are present in the generators [13, 25]. Each transposition is a unique cycle. On the other hand, randomly distributed automorphisms, as returned by Traces and dejavu, satisfy having a unique cycle with high probability, as argued above.

Unique Cycle To Cyclic Order. We now assume we are given an invariant unique cycle C. The idea is now to extend C using the generators S to a cycle which encompasses the entire orbit. Crucially, the extension ensures that the result is still invariant (i.e. if we do this for all orbits simultaneously, the final permutation will be invariant).

[1] In group theoretic terms, π is in $C_{\mathrm{Sym}(\Omega)}(C_{\mathrm{Sym}(\Omega)}(\langle s_1, \dots, s_m \rangle))$.

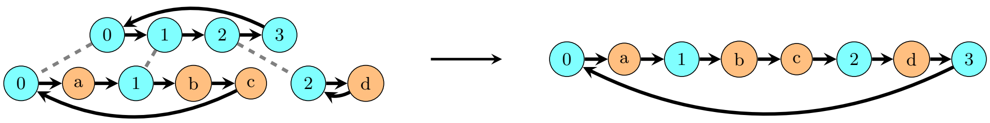


Figure 7 An illustration of the cycle overlap algorithm. Overlapping cycles $C = (0, 1, 2, 3)$ and $D = \{(0, a, 1, b, c), (2, d)\}$.

Algorithm 3 The cycle overlap algorithm.

```
function CycleOverlap($C$, $D$)
    Input : directed cycle $C$, overlapping pair-wise disjoint directed cycles
            $D = \{D_1, \ldots, D_m\}$ (with $\forall D_i \in D\colon D_i \cap C \neq \emptyset$,
            $\forall D_i, D_j \in D\colon i \neq j \implies D_i \cap D_j = \emptyset$)
    Output : directed cycle containing all vertices $C \cup D$
    $C' := \bigcup_{D_i \in D} C \cap D_i$;
    for ( $c \in C'$ )
        $D'$ := read $D$ from $c$ to next vertex of $C'$;
        insert $D'$ after $c$ in $C$;
    return $C$;
```

We now describe the *cycle overlap* algorithm, which gets as input a directed cycle C, as well as a collection of cycles $D_1, \ldots, D_m$ which must be pair-wise disjoint. Furthermore, each D_i must have one vertex in common with C. The result is a cycle C' that contains all vertices of all the cycles. A formal description can be found in Algorithm 3.

Description of Algorithm 3. The algorithm first checks which vertices of $D = D_1 \cup \cdots \cup D_m$ appear in C, and records them into the set C'. Then, for each $c \in C'$ in the overlap of C and D, the algorithm walks along the respective cycle D_i containing c, and records all vertices it observes into D'. It walks along the cycle until another $c' \in C'$ is reached (it may record the entire cycle D_i, i.e., $c' = c$ may hold). Finally, D' is inserted as a path into C.

The output of the process is invariant under the cyclic orders involved. This means no matter in which order the cycles from D are processed, the algorithm always returns the same cyclic order. Figure 7 illustrates the algorithm.

Runtime of Algorithm 3. We may use a doubly-linked list structure for directed cycles C and D, and an array A to link vertices V to their position in C in time $\mathcal{O}(1)$. Assuming these data structures, inserting a D' into C can be performed in time $|D'|$. Indeed, with these data structures, we can implement the entire algorithm in time $\mathcal{O}(\Sigma_{i \in \{1,\ldots,m\}}|D_i|)$. We may also update the array A to include the new vertices of C added from D.

To get a unique cyclic order, we repeatedly combine C with cycles appearing in generators that intersect C. Every cycle in a generator only has to be processed once. Eventually C contains the entire orbit. With careful management of usage-lists of vertices in cycles of generators, the overall algorithm can be implemented in instance-linear time.

7.3 Cyclic Order to Equivalent Orbits

We finally describe how to find equivalent orbits, assuming invariant cyclic orders are given on the orbits. An invariant cyclic order for the vertices of each orbit moves us one step closer to the orbits of the cycle type graph. There are however still many potential bijections between orbits: indeed, we do not know how each cyclic order should be rotated. We therefore describe a procedure to refine the cyclic order further.

Algorithm 4 High-level procedure to obtain equivalent orbits.

```
function EquivalentOrbits($S, \Delta_1, \dots, \Delta_m, C_{\Delta_1}, \dots, C_{\Delta_m}$)
    Input  : group $\langle S \rangle = \Gamma \leq S(\Omega)$, orbit partition $\Delta_1, \dots, \Delta_m$ of $\Gamma$, unique cycles
             $C_{\Delta_1}, \dots, C_{\Delta_m}$
    Output : partition of $v \in \Omega$ into equivalent orbits
    overlap each unique cycle $C_{\Delta_i}$ with $S$ to obtain canonical cycle $C'_{\Delta_i}$;
    construct enhanced cycle type graph $\mathcal{C}'(S)$;
    $\pi :=$ apply color refinement to $\mathcal{C}'(S)$;
    return $\pi$
```

We introduce the *enhanced cycle type graph* $\mathcal{C}'(S)$. We are provided an invariant cyclic order for each orbit Δ of Γ, which we denote by C_Δ. First, we add to the cycle type graph (Subsection 7.1) a cycle gadget for each C_Δ. As before, we color the cycle gadget C_Δ according to its cycle length. Next, we enhance all other cycle gadgets using distance information of C_Δ: in every cycle gadget we mark each directed edge $v_1 \rightarrow v_2$ with the length of the (directed) path from v_1 to v_2 in C_Δ (see Figure 6b). We write $v_1 \xrightarrow{c} v_2$ whenever the path from v_1 to v_2 in C_Δ has length c. Note that while we use edge-labels for clarity, these can be encoded back into vertex colors (see [19, Proof of Lemma 15]).

Just like with the cycle type graph, the automorphism group of the enhanced cycle type graph $\mathcal{C}'(S)$ is the centralizer of Γ and $(C_{\mathrm{Sym}(\Omega)}(\Gamma), \mathcal{C}'(\Gamma))$ is a joint graph/group pair (see Lemma 9). However, it is easier to compute the orbit partition of $\mathcal{C}'(S)$.

In fact, our method of obtaining the orbits of $\mathcal{C}'(S)$ is rather straightforward: we apply the color refinement procedure to the enhanced cycle type graph $\mathcal{C}'(S)$. The technical proof that color refinement indeed computes the orbit partition can be found in the full version [5]:

▶ **Lemma 11.** *Color refinement computes the orbit partition of the enhanced cycle type graph.*

Given our high-level procedure in Algorithm 4, and given that color refinement can be computed in quasi-linear time as previously discussed, leads to the following theorem:

▶ **Theorem 12.** *Given access to a unique cycle per orbit, there is an instance-quasi-linear algorithm which computes for a joint graph/group pair (G, S) a partition π of equivalent orbits. Given two equivalent orbits $\Delta_1 \equiv \Delta_2$, there is an algorithm which computes from π a corresponding matching $b \colon \Delta_1 \rightarrow \Delta_2$ such that for all $\varphi \in \Gamma$ and $\delta \in \Delta_1$, $\varphi(b(\delta)) = b(\varphi(\delta))$ in time $\mathcal{O}(|\Delta_1|)$.*

8 Conclusion and Future Work

Exploiting our concept of joint graph/group pairs, we proposed new, asymptotically faster algorithms for the SAT-symmetry interface. However, most of the new concepts and approaches of this paper do not only apply to the domain of SAT, but also for example to MIP [25] and CSP [14]. More computational tasks should be considered in this context, the most prominent one arguably being pointwise stabilizers [30].

Our new algorithms exploit subroutines with highly efficient implementations available, but otherwise do not use any complicated data structures. We intend to implement the algorithms and integrate them into the symmetry detection preprocessor SASSY [6].

Finally, in some classes of SAT instances, more complex symmetry structures may arise. Analyzing and taking advantage of these structures is potential future work. For example, in the pigeonhole principle, BreakID finds overlapping row interchangeability groups and breaks these groups partially. By virtue of being overlapping, the symmetry breaking constraints produced are *not* guaranteed to be complete. Specifically, the pigeonhole principle is an example of instances whose symmetries form a *wreath product* of two symmetric groups, i.e., S_n *wr* S_m (n pigeons, m holes). Procedures to detect and exploit such groups (e.g., by first using blocks of imprimitivity [30] followed by the algorithms of this paper) could be of practical interest.

References

1 Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. Shatter: efficient symmetry-breaking for boolean satisfiability. In *Proceedings of the 40th Design Automation Conference, DAC*, pages 836–839. ACM, 2003. doi:10.1145/775832.776042.

2 Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult SAT instances in the presence of symmetry. In *Proceedings of the 39th Design Automation Conference, DAC*, pages 731–736. ACM, 2002. doi:10.1145/513918.514102.

3 Markus Anders and Pascal Schweitzer. Parallel computation of combinatorial symmetries. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA*, volume 204 of *LIPIcs*, pages 6:1–6:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.6.

4 Markus Anders and Pascal Schweitzer. Search problems in trees with symmetries: Near optimal traversal strategies for individualization-refinement algorithms. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 198 of *LIPIcs*, pages 16:1–16:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.16.

5 Markus Anders, Pascal Schweitzer, and Mate Soos. Algorithms transcending the sat-symmetry interface. *CoRR*, abs/2306.00613, 2023. doi:10.48550/arXiv.2306.00613.

6 Markus Anders, Pascal Schweitzer, and Julian Stieß. Engineering a preprocessor for symmetry detection. *CoRR*, abs/2302.06351, 2023. doi:10.48550/arXiv.2302.06351.

7 Paul Beame, Richard M. Karp, Toniann Pitassi, and Michael E. Saks. The efficiency of resolution and davis–putnam procedures. *SIAM J. Comput.*, 31(4):1048–1075, 2002. doi:10.1137/S0097539700369156.

8 Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*, pages 3698–3707. AAAI Press, 2022. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20283.

9 Mun See Chang and Christopher Jefferson. Disjoint direct product decompositions of permutation groups. *J. Symb. Comput.*, 108:1–16, 2022. doi:10.1016/j.jsc.2021.04.003.

10 James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 148–159. Morgan Kaufmann, 1996.

11 Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for CNF. In Sharad Malik, Limor Fix, and Andrew B. Kahng, editors, *Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*, pages 530–534. ACM, 2004. doi:10.1145/996566.996712.

12 Jo Devriendt, Bart Bogaerts, and Maurice Bruynooghe. Symmetric explanation learning: Effective dynamic symmetry handling for SAT. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing – SAT*, volume 10491 of *LNCS*, pages 83–100. Springer, 2017. doi:10.1007/978-3-319-66263-3_6.

13 Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Improved static symmetry breaking for SAT. In Nadia Creignou and Daniel Le Berre, editors, *Theory and*

Applications of Satisfiability Testing – SAT, volume 9710 of *LNCS*, pages 104–122. Springer, 2016. doi:10.1007/978-3-319-40970-2_8.

14 Pierre Flener, Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson, and Toby Walsh. Breaking row and column symmetries in matrix models. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming – CP*, volume 2470 of *LNCS*, pages 462–476. Springer, 2002. doi:10.1007/3-540-46135-3_31.

15 The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.12.2*, 2022. URL: https://www.gap-system.org.

16 Solomon W. Golomb and Peter Gaal. On the number of permutations on n objects with greatest cycle length k. *Advances in Applied Mathematics*, 20(1):98–107, 1998. doi:10.1006/aama.1997.0567.

17 Andrew Grayland, Christopher Jefferson, Ian Miguel, and Colva M. Roney-Dougal. Minimal ordering constraints for some families of variable symmetries. *Annals of Mathematics and Artificial Intelligence*, 57:75–102, 2009.

18 Tommi A. Junttila and Petteri Kaski. Conflict propagation and component recursion for canonical labeling. In Alberto Marchetti-Spaccamela and Michael Segal, editors, *Theory and Practice of Algorithms in (Computer) Systems – First International ICST Conference, TAPAS*, volume 6595 of *LNCS*, pages 151–162. Springer, 2011. doi:10.1007/978-3-642-19754-3_16.

19 Sandra Kiefer, Ilia Ponomarenko, and Pascal Schweitzer. The weisfeiler-leman dimension of planar graphs is at most 3. *J. ACM*, 66(6):44:1–44:31, 2019. doi:10.1145/3333003.

20 Sandra Kiefer, Pascal Schweitzer, and Erkal Selman. Graphs identified by logics with counting. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS, Part I*, volume 9234 of *LNCS*, pages 319–330. Springer, 2015. doi:10.1007/978-3-662-48057-1_25.

21 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP*, volume 210 of *LIPIcs*, pages 34:1–34:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CP.2021.34.

22 Brendan D. McKay. Practical graph isomorphism. In *10th. Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, 1980)*, pages 45–87, 1981.

23 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

24 M. Neunhöffer, Á. Seress, and M. Horn. recog, a package for constructive recognition of permutation and matrix groups, Version 1.4.2. https://gap-packages.github.io/recog, September 2022. GAP package.

25 Marc E. Pfetsch and Thomas Rehn. A computational comparison of symmetry handling methods for mixed integer programs. *Math. Program. Comput.*, 11(1):37–93, 2019. doi:10.1007/s12532-018-0140-y.

26 Adolfo Piperno. Isomorphism test for digraphs with weighted edges. In Gianlorenzo D'Angelo, editor, *17th International Symposium on Experimental Algorithms, SEA*, volume 103 of *LIPIcs*, pages 30:1–30:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.SEA.2018.30.

27 Jean-Francois Puget. Symmetry breaking using stabilizers. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming – CP*, volume 2833 of *LNCS*, pages 585–599. Springer, 2003. doi:10.1007/978-3-540-45193-8_40.

28 Ashish Sabharwal. Symchaff: exploiting symmetry in a structure-aware satisfiability solver. *Constraints An Int. J.*, 14(4):478–505, 2009. doi:10.1007/s10601-008-9060-1.

29 Karem A. Sakallah. Symmetry and satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 509–570. IOS Press, 2021. doi:10.3233/FAIA200996.

30 Ákos Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003. doi:10.1017/CBO9780511546549.

Proof Complexity of Propositional Model Counting

Olaf Beyersdorff
Friedrich-Schiller-Universität Jena, Germany

Tim Hoffmann
Friedrich-Schiller-Universität Jena, Germany

Luc Nicolas Spachmann
Friedrich-Schiller-Universität Jena, Germany

Abstract

Recently, the proof system MICE for the model counting problem #SAT was introduced by Fichte, Hecher and Roland (SAT'22). As demonstrated by Fichte et al., the system MICE can be used for proof logging for state-of-the-art #SAT solvers.

We perform a proof-complexity study of MICE. For this we first simplify the rules of MICE and obtain a calculus MICE′ that is polynomially equivalent to MICE. Our main result establishes an exponential lower bound for the number of proof steps in MICE′ (and hence also in MICE) for a specific family of CNFs.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases model counting, #SAT, proof complexity, proof systems, lower bounds

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.2

Funding *Olaf Beyersdorff*: Carl-Zeiss Foundation and DFG grant BE 4209/3-1.
Tim Hoffmann: Carl-Zeiss Foundation.

1 Introduction

The problem to decide whether a Boolean formula is satisfiable (SAT) is one of central problems in computer science, both theoretically and practically. From the theoretical side, SAT is the canonical NP-complete problem [14], making it intractable unless P=NP. From the practical side, the "SAT revolution" [31] with the evolution of practical SAT solvers has turned SAT into a tractable problem for many industrial instances [5].

In this paper we consider the *model counting problem* (#SAT) which asks how many satisfying assignments a given Boolean formula has. While #SAT is obviously a generalization of SAT, it is presumably much harder. #SAT is the canonical complete problem for the function class #P. While FP=#P would imply P=NP, it is known that FP=#P is even equivalent to P=PP. The power of #SAT is also illustrated by Toda's theorem [30] stating that any problem in the polynomial hierarchy can be solved in polynomial time with oracle access to #SAT.

Despite its higher complexity, #SAT solving has been actively pursued through the past two decades [20] and a number of #SAT solvers have been developed throughout the years. In fact, the past years have witnessed increased interest in #SAT solving with an annual model counting competition being organised since 2020 as part of the SAT conference [17]. #SAT solvers allow to tackle a large variety of real-world questions, including all kinds of problems in the areas probabilistic reasoning [2, 25], risk analysis [16, 34] and explainable artificial intelligence [3, 28].

Unlike in SAT solving where conflict-driven clause learning (CDCL) [26] dominates the scene, there are a number of conceptually different approaches to #SAT solving, including the lifting of standard techniques from SAT-solving [29], employing knowledge compilation [24], and via dynamic programming [19]. While some approaches try to approximate the number of solutions, we will only consider exact model counting in the following.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 2; pp. 2:1–2:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There is a tight correspondence between practical SAT solving and propositional proof systems [9]. While we know that in principle every SAT solver implicitly defines a proof system, a seminal result of [1, 27] established that CDCL (at least in its nondeterministic version) is equivalent to the resolution proof system. However, practical CDCL with e.g. the VSIDS heuristics corresponds to an exponentially weaker proof system than resolution [32]. In the same vein, there has recently been a line of research to understand the correspondence between solvers for quantified Boolean formulas (QBF) and QBF resolution proof systems [4, 6, 7].

This correspondence between solvers and proofs is not only of theoretical, but also of immense practical interest as it can be used for *proof logging*, i.e. for certifying the correctness of solvers on unsatisfiable SAT or QBF instances. Optimised proof systems have been devised in terms of RAT/DRAT for SAT [22, 33] and QRAT for QBF [23] for this purpose. These proof systems aim to capture all modern solving techniques, including preprocessing and therefore tend to be very powerful [10, 13]. In particular, in contrast to weak proof systems such as resolution, no lower bounds are known for RAT or QRAT.

In sharp contrast, far less is known about the correspondence of model counting solvers to proof systems. To our knowledge, there are currently two proof systems for #SAT. One is a static proof system based on decision DNNFs called kcps(#SAT) (the acronym stands for Knowledge Compilation based Proof System for #SAT) [11]. The other, a line based proof system called MICE [18] (the acronym stands for Model-counting Induction by Claim Extension), was just introduced at the last SAT conference [18]. Interestingly, the system MICE not only provides a theoretical proof system for #SAT, but also allows proof logging for a number of state-of-the-art solvers in model counting, including sharpSAT [29], DPDB [19] and D4 [24], as demonstrated in [18]. Hence MICE proofs can be used to verify the correctness of answers of these #SAT solvers.

1.1 Our Contributions

We perform a proof complexity analysis of the #SAT proof system MICE from [18]. Prior to this paper, no proof complexity results for MICE were known. Our results can be summarised as follows.

(a) A simplified proof system MICE'. We analyse the proof system MICE and define a somewhat simplified calculus MICE$'$. Lines in MICE are of the form $((F, V), A, c)$ where F is a propositional formula V is a set of variables, A is a partial assignment and $c \in \mathbb{N}$. Semantically, these lines express that the formula F under the partial assignment A has precisely c models. The system MICE then employs four rules to derive new lines with the ultimate goal to derive a line $((F, \mathsf{vars}(F)), \emptyset, c)$. Thus in the ultimate line, c is the number of models of the formula F.

The four rules of the system include one axiom rule for satisfying total assignments and three rules to compose, join and extend existing lines. All the rules have a rather extensive set of side conditions to verify their applicability. For the composition rule this even includes an external resolution proof to check that the composition of claims in the rule indeed covers all models.

The variable set V does not feature in the semantical explanation above. While it might be tempting to choose $V = \mathsf{vars}(F)$ for all lines (as is done in the final claim), we show that this restriction is too strong and results in an exponentially weaker system. Nevertheless, we show that we can slightly adapt the rules of MICE (in particular the extension rule) and obtain a system MICE$'$ for which we can impose $V = \mathsf{vars}(F)$ for all lines without weakening the system. Lines in MICE$'$ therefore can take the form (F, A, c). This allows

allows to eliminate and simplify some of the side conditions for the original rules of MICE when transferring to MICE′. Our simplified system MICE′ is as strong as MICE in terms of simulations (Propositions 16 and 17). Hence also MICE′ can be used for proof logging for the #SAT solvers mentioned above.

(b) Lower bounds for MICE and MICE′. In our main result we show an exponential lower bound for the proof size in MICE′ (and hence also for MICE) for a specific family of CNFs.

As mentioned above, the composition rule of MICE (and MICE′) incorporates resolution proofs. Exploiting this feature, it is not too hard to transfer resolution lower bounds to MICE′. In fact, we can show that on unsatisfiable formulas, resolution is polynomially equivalent to MICE′ (Theorem 18).

However, we would view such a transferred resolution lower bound not as a "genuine" and interesting lower bound for MICE′. We therefore show a stronger bound for MICE′ for *the number of proof steps* (where we disregard the size of the attached resolution proofs). In our main result we show a lower bound of $2^{\Omega(n)}$ for the number of proof steps for a specific set of CNFs, termed $\mathsf{XOR\text{-}PAIRS}_n$, based on the parity function (Theorem 23). Technically, our lower bound is established by showing that in MICE′ proofs of $\mathsf{XOR\text{-}PAIRS}_n$, all applications of the join and extension rules preserve the model count.

1.2 Relations to DNNFs

One of the anonymous reviewers highlighted that there is a close connection between our work here and Decomposable Negation Normal Forms (DNNFs) as investigated in [8, 11, 12]. We were not aware of that work and would like to thank the reviewer for pointing that out.

In particular, it appears that from a MICE′ proof a decision DNNF can be efficiently extracted. Hence, alternatively to our directly obtained lower bound for MICE′ in Section 5, one could employ decision DNNF lower bounds as shown via communication complexity in [8] for MICE′ lower bounds.

1.3 Organisation

The remainder of this paper is organised as follows. After reviewing some standard notions from propositional logic and proof systems in Section 2, we revise the #SAT proof system MICE from [18] in Section 3 and show some properties of the system. This gives rise to a simplified proof system MICE′ which we define in Section 4. Section 5 contains our main results on the exponential lower bound for MICE′ (and hence for MICE). We conclude in Section 6 with relations to some open questions and future directions.

2 Preliminaries

We introduce some notations used in this paper. A literal l is a variable z or its negation $\overline{z}$, with $\mathsf{var}(l) = z$. A clause is a disjunction of literals, a conjunctive normal form (CNF) is a conjunction of clauses. Often, we write clauses as sets of literals and formulas as sets of clauses. We assume that every propositional formula is written in CNF.

For a formula F, $\mathsf{vars}(F)$ denotes the set of all variables that occur in F, and $\mathsf{lits}(F)$ is the set of all literals of F. If $C \in F$ is a clause and $V \subseteq \mathsf{vars}(F)$ is a set of variables, we define $C|_V = \{l \in C \mid \mathsf{vars}(l) \in V\}$ and $F|_V$ denotes the formula F with every clause C replaced by $C|_V$. An assignment is a function α mapping variables to Boolean values. If a function F evaluates to true under an assignment α, we say α satisfies F and write $\alpha \models F$. We also allow α to be a partial assignment to $\mathsf{vars}(F)$ or to contain variables not occurring in F. Occasionally, we interpret an assignment as a CNF consisting of precisely those unit clauses that specify the assignment. Therefore, the set operations are well defined for formulas and assignments. We say that two assignments are consistent if their union is satisfiable. For some set of variables X, $\langle X\rangle$ denotes the set of all $2^{|X|}$ possible assignments to X.

In this paper we are interested in proof systems as introduced in [15]. Formally, a proof system for a language L is a polynomial-time computable function f with $\mathsf{rng}(f) = L$. If $f(w) = x$, then w is called f-proof of $x \in L$. In order to compare proof systems we need the notion of simulations. Let f and g be proof systems for language L. We say that f simulates g, if for any g-proof w there exists an f-proof w' with $|w'| = |w|^{O(1)}$ and $f(w') = g(w)$. If we can compute w' in polynomial time from w, we say that f p-simulates g. Two proof systems are (p-)equivalent if they (p-)simulate each other.

For the language UNSAT of unsatisfiable CNFs, resolution is arguably the most studied proof system. It operates on Boolean formulas in CNF and has only one rule. This resolution rule can derive $C \cup D$ from $C \cup \{x\}$ and $D \cup \{\overline{x}\}$ with arbitrary clauses C, D and variable x. A resolution refutation of a CNF is a derivation of the empty clause $\Box$. We sometimes add a weakening rule that enables us to derive $C \cup D$ from C for arbitrary clauses C and D. However, it is well-known that any resolution refutation that uses weakening can be efficiently transformed into a resolution refutation without weakening.

3 The Proof System MICE for #SAT

In this section we recall the MICE proof system for #SAT from [18] and show some basic properties of the system.

▶ **Definition 1** ([18]). *A* claim *is a triple* $((F, V), A, c)$ *where* F *is a propositional formula in CNF,* V *is a set of variables,* A *is an assignment with* $\mathsf{vars}(A) \subseteq V$ *and* $c \in \mathbb{N}$*. For such a claim, let* $\mathsf{Mod}_A(F, V) := \{\alpha \in \langle V\rangle \mid \alpha \models F \cup A\}$*. The claim is* correct *if* $c = |\mathsf{Mod}_A(F, V)|$.

Claims will be the lines in our proof systems for model counting. Semantically, they describe that the formula F under the partial assignment A has exactly c models. The partial assignment A is sometimes also referred to as the assumption. What is perhaps a bit mysterious at this point is the role of the variable set V. We will get to this shortly.

The rules of MICE are Exactly One Model (1-Mod), Composition (Comp), Join (Join) and Extension (Ext). They are specified in Figure 1. We give some intuition on the rules. The axiom rule (1-Mod) states that if a complete assignment A satisfies a formula F, then F has exactly one model under A.

With (Comp) we can sum up model counts of a formula F under different partial assignments $A_1, \ldots, A_n$ in order to weaken the assumption to a partial assignment A. This is only sound if the solutions of F under assumptions $A_1, \ldots, A_n$ form a disjoint partition of the full solution space of F under A. That this is indeed the case can be verified with an independent proof, e.g. in propositional resolution. This proof is called an *absence of models statement.*

The (Join) rule allows us to multiply the model counts of two formulas that are completely independent restricted to the assumptions. Finally, with (Ext), we can extend simultaneously all models, i.e. we enlarge the formula and assumption without changing the count.

We can now formally define MICE proofs.

▶ **Definition 2** (Fichte, Hecher, Roland [18]). *A* MICE trace *is a sequence* $\pi = (I_1, \ldots, I_k)$ *where for each* $i \in [k]$*, either*

- *I_i is a claim if I_i is derived by one of (1-Mod), (Join), (Ext) or*
- *$I_i = (I, \rho)$ if the claim I is derived by (Comp) and ρ is the resolution refutation for the respective absence of models statement.*

A MICE proof *of a formula φ is a* MICE *trace $\pi = (I_1, \ldots, I_k)$ where I_k is (or contains in case of (Comp)) the claim $((\varphi, \mathsf{vars}(\varphi)), \emptyset, c)$ for some $c \in \mathbb{N}$.*

In [18] it is shown that MICE is a sound and complete proof system for #SAT.

For measuring the *proof size*, we use two natural options. $s(\pi)$ notates the size of π which is the total number of claims plus the number of clauses in resolution proofs in the absence of models statements. $c(\pi)$ counts only the number of claims a proof has which is exactly the number of inference steps that the proof needs.

Exactly One Model.

$$\frac{}{((F,V),A,1)} \qquad \text{(1-Mod)}$$

- (O-1) $\mathsf{vars}(A) = V$,
- (O-2) A satisfies F.

Composition.

$$\frac{((F,V),A_1,c_1),\ldots,((F,V),A_n,c_n)}{((F,V),A,\sum_{i\in[n]} c_i)} \qquad \text{(Comp)}$$

- (C-1) $\mathsf{vars}(A_1) = \mathsf{vars}(A_2) = \cdots = \mathsf{vars}(A_n)$ and $A_i \neq A_j$ for $i \neq j$,
- (C-2) $A \subseteq A_i$ for all $i \in [n]$,
- (C-3) there exists a resolution refutation of $A \cup \{C|_V \mid C \in F\} \cup \{\overline{A_i} \mid i \in [n]\}$. Such a refutation is included into the trace and is called an *absence of models statement.*

Join.

$$\frac{((F_1,V_1),A_1,c_1),((F_2,V_2),A_2,c_2)}{((F_1 \cup F_2, V_1 \cup V_2), A_1 \cup A_2, c_1 \cdot c_2)} \qquad \text{(Join)}$$

- (J-1) A_1 and A_2 are consistent,
- (J-2) $V_1 \cap V_2 \subseteq \mathsf{vars}(A_i)$ for $i \in \{1,2\}$,
- (J-3) $\mathsf{vars}(F_i) \cap ((V_1 \cup V_2) \setminus V_i) = \emptyset$ for $i \in \{1,2\}$.

Extension.

$$\frac{((F_1,V_1),A_1,c)}{((F,V),A,c)} \qquad \text{(Ext)}$$

- (E-1) $F_1 \subseteq F$, $V_1 \subseteq V$,
- (E-2) $V \setminus V_1 \subseteq \mathsf{vars}(A)$,
- (E-3) $A|_{V_1} = A_1$,
- (E-4) A satisfies $F \setminus F_1$,
- (E-5) for every $C \in F_1$: $A|_{V \setminus V_1}$ does not satisfy C.

Figure 1 Inference rules for MICE [18].

In a correct claim $((F,V),A,c)$ the count c is uniquely determined by the the formula F, set of variables V and assumption A. Therefore, we often omit c and refer to the claim as $((F,V),A)$. To ease notation we will usually just write a MICE proof as as sequence of claims $I_1, \dots, I_m$ and do not explicitly record the used absence of models statements. We just assume that whenever we use (Comp), the necessary resolution refutation is part of the MICE proof.

If a formula F is satisfied by the partial assignment A, we can set the remaining variables arbitrarily. Therefore, the component $(F, \mathsf{vars}(F))$ has exactly $2^{|\mathsf{vars}(F)|-|\mathsf{vars}(A)|}$ models under assumption A. The following construction shows that we can efficiently derive the corresponding claim in MICE.

▶ **Proposition 3.** *If some assumption A satisfies an arbitrary formula F, there is a* MICE *derivation of the claim $I = ((F, \mathsf{vars}(F)), A, 2^{|\mathsf{vars}(F) \setminus \mathsf{vars}(A)|})$ with $s(\pi) = 7 \cdot (|\mathsf{vars}(F) \setminus \mathsf{vars}(A)|)$ and $c(\pi) = 4 \cdot (|\mathsf{vars}(F) \setminus \mathsf{vars}(A)|)$.*

Proof. Let $\mathsf{vars}(F) \setminus \mathsf{vars}(A) = \{x_1, \dots, x_n\}$. For every $i \in [n]$ we derive $I_i^1 = ((\emptyset, \mathsf{vars}(A) \cup \{x_i\}), A \cup \{x_i\}, 1)$ and $I_i^0 = ((\emptyset, \mathsf{vars}(A) \cup \{x_i\}), A \cup \{\overline{x}_i\}, 1)$ with (1-Mod). This is possible since every assignment satisfies the empty formula. With (Comp) we get $I_i = ((\emptyset, \mathsf{vars}(A) \cup \{x_i\}), A, 2)$ using the absence of models statement $\rho_i = ((x_i), (\overline{x}_i), \Box)$. We use (Join) of I_1 and I_2, then (Join) of the result and I_3, and so on. The requirements (J-1), (J-2), and (J-3) are satisfied. In this way we get $((\emptyset, \mathsf{vars}(F)), A, 2^{|\mathsf{vars}(F) \setminus \mathsf{vars}(A)|})$. We use (Ext) to obtain $I = ((F, \mathsf{vars}(F)), A, 2^{|\mathsf{vars}(F) \setminus \mathsf{vars}(A)|})$. It is easy to see that all requirements (E-1) to (E-5) are satisfied. For (E-4), we use that A satisfies F. In total we use $4n$ MICE steps to derive I and we have n absence of models statements with 3 clauses each. ◀

We investigate some properties that any claim in a MICE proof has to fulfill. We assume that any MICE proof has no redundant claims, i.e. in the corresponding proof dag, there is a path from any node to the final claim. We also observe that for all inference rules, the derived F and V never shrink. This leads to the following two observations:

▶ **Observation 4.** *If $((F, V), A)$ is derived from $((F_1, V_1), A_1)$ in a* MICE *trace (not necessarily in one step), then $F_1 \subseteq F$ and $V_1 \subseteq V$.*

Therefore, any claim $((F, V), A)$ in a MICE *proof of φ fulfills $F \subseteq \varphi$ and $V \subseteq \mathsf{vars}(\varphi)$.*

From Definition 1 it is not obvious how F and V are related. Intuitively, one might be tempted to set $V = \mathsf{vars}(F)$ for any claim $((F, V), A)$. However, this would make the proof system exponentially weaker as we will see later. Lemma 6 will show that we can at least assume $\mathsf{vars}(F) \subseteq V$ for every claim. To show this we need the following lemma:

▶ **Lemma 5.** *For any claim $((F, V), A)$ and any variable x, if $x \in \mathsf{vars}(F) \setminus V$, then literals x and $\overline{x}$ cannot both occur in F.*

Proof Sketch. Suppose there exists such an x. Since $((F, V), A)$ is not redundant, there is a path to the final claim. Thus, there have to be claims $((F_1, V_1), A_1)$ and $((F_2, V_2), A_2)$ directly adjacent in the path with $F \subseteq F_1 \subseteq F_2$, $V \subseteq V_1 \subseteq V_2$ and $x \notin V_1$, $x \in V_2$. Now $((F_2, V_2), A_2)$ is directly derived from $((F_1, V_1), A_1)$ in one step. We can argue that this is not possible. ◀

▶ **Lemma 6.** *Let a formula φ and a* MICE *proof π for φ be given. Then there is a* MICE *proof π' satisfying $\mathsf{vars}(F) \subseteq V$ for any claim $((F, V), A) \in \pi'$ such that $s(\pi') = O(s(\pi)^3)$ and $c(\pi') = c(\pi)$.*

Proof Sketch. Let $\pi = (I_1, \dots, I_m)$ with $I_i = ((F_i, V_i), A_i)$. Because of Lemma 5, for any $i \in [m]$, we can assume that there is no variable $x \in \mathsf{vars}(F_i) \setminus V_i$ that occurs in both polarities in F_i. Let $\alpha_i \in \langle \mathsf{vars}(F_i) \setminus V_i \rangle$ be the assignment that does not satisfy any clause in F_i, i.e. if x is in F_i we assign $\alpha_i(x) = 0$ and vice versa. For every claim I_i, α_i exists and it is unique. We define

$$f(((F_i, V_i), A_i)) := ((F_i, V_i \cup \mathsf{vars}(F_i)), A_i \cup \alpha_i)$$

with the unique α_i defined above. We show by induction that $(f(I_1), \ldots, f(I_m))$ is a valid MICE proof for φ. ◀

In the following we always assume $\mathsf{vars}(F) \subseteq V$ for any claim $((F, V), A)$. With this requirement, the conditions of the inference rules can be simplified.

▶ **Corollary 7.** *If we require* $\mathsf{vars}(F) \subseteq V$ *for every claim* $((F, V), A)$*, the following simplifications for the* MICE *rules apply:*

- *We can simplify the absence of models statement in the requirement (C-2) to be a refutation of* $F \cup A \cup \{\overline{A_i} \mid i \in [n]\}$.
- *We can remove condition (J-3) for (*Join*).*
- *We can remove condition (E-5) for (*Ext*).*

However, imposing the stronger condition $\mathsf{vars}(F) = V$ for every claim $((F, V), A)$ would make the proof system exponentially weaker as we illustrate with the next proposition.

▶ **Lemma 8.** *There is a family of formulas* $(T_n)_{n\in\mathbb{N}}$ *such that for both measures* $s(\cdot)$ *and* $c(\cdot)$ *holds:*

- T_n *has polynomial-size* MICE *proofs and*
- *if* $\mathsf{vars}(F) = V$ *is required for all claims* $((F, V), A)$*, the shortest* MICE *proof of* T_n *has exponential size.*

Proof Sketch. Consider the formula T_n that only has one clause $(x_1 \vee x_2 \vee \cdots \vee x_n)$.

To construct a polynomial-size MICE proof, we derive $((\emptyset, \mathsf{vars}(T_n)), \{x_1 = 1\}, 2^{n-1})$ with a small number of applications of (1-Mod) and (Join). We get $((T_n, \mathsf{vars}(T_n)), \{x_1 = 1\}, 2^{n-1})$ with (Ext). Similarly, we derive $((T_n, \mathsf{vars}(T_n)), \{x_1 = 0, x_2 = 1\}, 2^{n-2})$ and so on. With applications of (Comp) we combine these claims to $((T_n, \mathsf{vars}(T_n)), \emptyset, 2^n - 1)$.

We can show that any MICE proof with the additional requirement $\mathsf{vars}(F) = V$ needs to have a claim $((T_n, \mathsf{vars}(T_n)), \alpha)$ for every model $\alpha \in \mathsf{Mod}(T_n)$. Since T_n has $2^n - 1$ models, the proof has size $2^{\Omega(n)}$. ◀

4 A Simplified Proof System MICE' for #SAT

We now adapt MICE to a new proof system MICE′ that is as strong as MICE and only uses claims $((F, V), A)$ with components satisfying $V = \mathsf{vars}(F)$. Therefore, we can drop the explicit mentioning of the variable set V and only need to specify the formula F. This makes the resulting proof system more intuitive and easier to investigate for lower bounds.

The rules of MICE′ are Axiom (Ax), Composition (Comp'), Join (Join') and Extension (Ext'). They are specified in Figure 2.

The intuition for the rules (Comp') and (Join') are very similar to (Comp) and (Join) from MICE. The (Ax) rule enables us to derive the claim $(\emptyset, \emptyset, 1)$ which is trivially true. (Ext') is also similar to (Ext) with one important difference: If we use (Ext) in MICE, the assumption has to assign all variables that are added to the claim. As result, we extend one model of the original claim to one new model. In (Ext') however, this is not necessarily the case. As long as the new assumption satisfies all added clauses, we are allowed to leave new introduced variables unassigned in the assumption. Like this we extend every model of the original claim to a set of new models, one for every possible assignment of these unassigned variables.

Axiom.

$$\frac{}{(\emptyset, \emptyset, 1)} \qquad \text{(Ax)}$$

Composition.

$$\frac{(F, A_1, c_1), \dots, (F, A_n, c_n)}{(F, A, \sum_{i \in [n]} c_i)} \qquad \text{(Comp')}$$

- (C-1) $\mathsf{vars}(A_1) = \mathsf{vars}(A_2) = \dots = \mathsf{vars}(A_n)$ and $A_i \neq A_j$ for $i \neq j$,
- (C-2) $A \subseteq A_i$ for all $i \in [n]$,
- (C-3) there exists a resolution refutation of $A \cup F \cup \{\overline{A_i} \mid i \in [n]\}$. Such a refutation is included into the trace and is called an *absence of models statement.*

Join.

$$\frac{(F_1, A_1, c_1), (F_2, A_2, c_2)}{(F_1 \cup F_2, A_1 \cup A_2, c_1 \cdot c_2)} \qquad \text{(Join')}$$

- (J-1) A_1 and A_2 are consistent,
- (J-2) $\mathsf{vars}(F_1) \cap \mathsf{vars}(F_2) \subseteq \mathsf{vars}(A_i)$ for $i \in \{1, 2\}$.

Extension.

$$\frac{(F_1, A_1, c_1)}{(F, A, c_1 \cdot 2^{|\mathsf{vars}(F) \setminus (\mathsf{vars}(F_1) \cup \mathsf{vars}(A))|})} \qquad \text{(Ext')}$$

- (E-1) $F_1 \subseteq F$,
- (E-2) $A|_{\mathsf{vars}(F_1)} = A_1$,
- (E-3) A satisfies $F \setminus F_1$.

Figure 2 Inference rules for MICE′.

▶ **Definition 9** (Adapted Proof System MICE′). *A* claim *is a triple* (F, A, c) *with* $\mathsf{vars}(A) \subseteq \mathsf{vars}(F)$*. For such a claim, let* $\mathsf{Mod}_A(F) := \{\alpha \in \langle \mathsf{vars}(F) \rangle \mid \alpha \models F \cup A\}$*. The claim is* correct *if* $c = |\mathsf{Mod}_A(F)|$*. The rules of* MICE′ *are (Ax), (Comp′), (Join′) and (Ext′). The notions of* MICE′ traces *and* MICE′ proofs *are defined analogously as for* MICE*. Furthermore, we use the same two measures for the proof size* $s(\cdot)$ *and* $c(\cdot)$*.*

As in the MICE proof system we often omit the count c of claims and assume that no redundant claims exist in MICE′ proofs, i.e. all claims are connected to the final claim.

We prove that all four derivation rules are sound, i.e. for every derived claim (F, A, c) holds $c = |\mathsf{Mod}_A(F)|$. In doing so, we will also provide some intuition on the semantic meaning of the rules.

▶ **Lemma 10.** *The inference rules of* MICE′ *are sound.*

Proof Sketch. To prove the soundness of every MICE′ rule, we associate every claim (F, A, c) with the set containing exactly the c models in $\mathsf{Mod}_A(F)$. With this interpretation, we can specify how every rule modifies these models. This way, we can show that the resulting model count is indeed correct for every MICE′ rule.

The soundness of (Ax) is obvious, since $\mathsf{Mod}_\emptyset(\emptyset) = \{\emptyset\}$.

To show soundness of (Comp'), let $(F, A, \sum_{i\in[n]} c_i)$ be derived with (Comp') from correct claims $(F, A_1, c_1), \ldots, (F, A_n, c_n)$. Then we can show

$$\mathsf{Mod}_A(F) = \{\alpha \in \langle \mathsf{vars}(F)\rangle \mid \alpha \models F \cup A\}.$$

Next, we show soundness of (Join'). For this, let $(F_1 \cup F_2, A_1 \cup A_2, c_1 \cdot c_2)$ be derived with (Join') from correct claims (F_1, A_1, c_1) and (F_2, A_2, c_2). We can show that

$$\mathsf{Mod}_{A_1\cup A_2}(F_1 \cup F_2) = \{\alpha_1 \cup \alpha_2 \mid \alpha_1 \in \mathsf{Mod}_{A_1}(F_1), \alpha_2 \in \mathsf{Mod}_{A_2}(F_2)\}.$$

Finally we have to show that (Ext') is sound. Assume (F, A, c) is derived with (Ext') from the correct claim (F_1, A_1, c_1). We can show

$$\mathsf{Mod}_A(F) = \{\alpha \cup (A \setminus A_1) \cup \beta \mid \alpha \in \mathsf{Mod}_{A_1}(F_1), \beta \in \langle \mathsf{vars}(F) \setminus (\mathsf{vars}(F_1) \cup \mathsf{vars}(A))\rangle\}.$$

Therefore, claims derived with MICE′ are correct. ◀

▶ **Corollary 11.** *Let claim $I = (F, A)$ and a model $\alpha \in \mathsf{Mod}_A(F)$ be given.*

- *If I is derived with (Comp') using claims $(F, A_1), \ldots, (F, A_n)$, then there exists exactly one $i \in [n]$ such that $\alpha \in \mathsf{Mod}_{A_i}(F_i)$.*
- *If I is derived with (Join') using claims (F_1, A_1) and (F_2, A_2), then for both $i \in [2]$ we have $\alpha|_{\mathsf{vars}(F_i)} \in \mathsf{Mod}_{A_i}(F_i)$.*
- *If I is derived with (Ext') using claim (F_1, A_1), then $\alpha|_{\mathsf{vars}(F_1)} \in \mathsf{Mod}_{A_1}(F_1)$.*

We introduce an additional rule (SA) which is similar to the construction in Proposition 3.

▶ **Definition 12** (Satisfying Assumption Rule). *Under the condition (S-1): A satisfies F, we allow to derive*

$$\frac{}{(F, A, 2^{|\mathsf{vars}(F)\setminus\mathsf{vars}(A)|})} \quad (SA).$$

This rule is sound and does not make MICE′ proofs much shorter.

▶ **Lemma 13.** *(SA) is sound. Further, if formula φ has a MICE′ proof π that can use the additional rule (SA), then there exists a MICE′ proof π' of φ with $s(\pi') = s(\pi) + 1$ and $c(\pi') = c(\pi) + 1$.*

Proof. Assume that we applied (SA) in π to derive claim $I = (F, A, 2^{|\mathsf{vars}(F)\setminus\mathsf{vars}(A)|})$. Then we can derive I without (SA) with two MICE′ steps in the following way. We use (Ax) to get $(\emptyset, \emptyset, 1)$ and then (Ext') to derive I. It is easy to see that conditions (E-1) and (E-2) are fulfilled. (E-3) follows directly from (S-1). The resulting counts are the same since $1 \cdot 2^{|\mathsf{vars}(F)\setminus(\mathsf{vars}(F_1))\cup\mathsf{vars}(A))|} = 2^{|\mathsf{vars}(F)\setminus\mathsf{vars}(A)|}$. Since we can simulate (SA) with the other sound MICE′ rules, (SA) is sound as well. If we replace all applications of (SA) like this, then the proof size increases at most by one, as we need (Ax) only once in the proof. ◀

To justify our definition of MICE′ we have to show that it is indeed a proof system for #SAT.

▶ **Theorem 14.** MICE′ *is a sound and complete proof system for #SAT.*

Proof. The soundness of MICE′ follows directly from the soundness of the inference rules as shown in Lemma 10.

Next, we show that MICE' is complete. For this, let an arbitrary formula φ be given. We can derive $I_\alpha = (\varphi, \alpha, 1)$ for every $\alpha \in \mathsf{Mod}(\varphi)$ with (SA). For all these models together there is an absence of models statement. Therefore, we can derive $(\varphi, \emptyset, |\mathsf{Mod}(\varphi)|)$ with (Comp') from all claims I_α. Note that for unsatisfiable formulas we can derive the final claim with a single application of (Comp').

In proof systems, it is also necessary that proofs can be verified in polynomial time. This is possible in MICE' since all conditions (C-1), (C-2), (C-3), (J-1), (J-2), (E-1), (E-2) and (E-3) are easy to check in polynomial time. ◀

Next, we show some basic properties of MICE'.

▶ **Lemma 15.** *Let claim (F_1, A_1) be used to derive (F, A) (not necessarily in one step). Then*
- $F_1 \subseteq F$,
- *if $x \in \mathsf{vars}(F_1) \cap \mathsf{vars}(A)$, then $x \in \mathsf{vars}(A_1)$ and $A(x) = A_1(x)$.*

Proof. Because every MICE' rule does not decrease the formula F, the first property is obvious.

Let $((F_1, A_1), \ldots, (F_n, A_n) = (F, A))$ be a path in this derivation. It is easy to check that for all four inference rules of MICE' we have $A_{i+1}|_{\mathsf{vars}(F_i)} \subseteq A_i$ for $i \in [n-1]$. We can restrict both sides and get

$$(A_{i+1}|_{\mathsf{vars}(F_i)})|_{\mathsf{vars}(F_1)} = A_{i+1}|_{\mathsf{vars}(F_i) \cap \mathsf{vars}(F_1)} = A_{i+1}|_{\mathsf{vars}(F_1)} \subseteq A_i|_{\mathsf{vars}(F_1)}.$$

Therefore,

$$A|_{\mathsf{vars}(F_1)} = A_n|_{\mathsf{vars}(F_1)} \subseteq A_{n-1}|_{\mathsf{vars}(F_1)} \subseteq \cdots \subseteq A_1|_{\mathsf{vars}}(F_1) = A_1.$$

From $A|_{\mathsf{vars}(F_1)} \subseteq A_1$ the second property follows. ◀

Using these properties, we can show that the new proof system MICE' is polynomially equivalent to MICE. Note that this result is true for both measures of proof size $s(\cdot)$ and $c(\cdot)$. To prove this equivalence, we show both simulations separately.

First we show that MICE' is at least as strong as MICE. This simulation is the more important one for this paper as it implies that lower bounds for MICE' do also apply for MICE.

▶ **Proposition 16.** MICE' *p-simulates* MICE.

Proof Sketch. Let $\pi = (I_1, \ldots, I_m)$ be a MICE proof of a given formula φ. We assume that $\mathsf{vars}(F) \subseteq V$ for all claims $((F, V), A)$ in π which is justified by Lemma 6. We can show by induction that for $f(((F, V), A)) := (F, A|_{\mathsf{vars}(F)})$ the sequence $\pi' = (f(I_1), \ldots, f(I_m))$ is a correct MICE' proof of φ. ◀

Next we show that MICE' is not stronger than MICE. Although this result is not needed for the lower bounds, it is nice to know how our new proof system MICE' relates to MICE exactly.

▶ **Proposition 17.** MICE *p-simulates* MICE'.

Proof Sketch. Let $\pi = I_1, \ldots, I_n$ with $I_i = (F_i, A_i)$ be a MICE' proof of a given formula φ. We define $f(I_i) := ((F_i, \mathsf{vars}(F_i)), A_i)$ and show that we can derive $f(I_k)$ using $f(I_1), \ldots, f(I_{k-1})$ with $O(|\mathsf{vars}(\varphi)|)$ MICE steps. ◀

5 Lower Bounds for MICE and MICE'

In this section we investigate the proof complexity of MICE'. For the analysis we use the two different measures of proof size.

First, we consider the proof size $s(\cdot)$. For that, we can easily lift known lower bounds from propositional resolution and get families of formulas that require MICE' proofs of exponential size.

However, one could argue, that this is not the kind of hardness we are interested in. In the second part we get a stronger result by showing a lower bound for the number of inference steps $c(\cdot)$, i.e. we ignore the sizes of the absence of models statements.

5.1 Lower Bounds for the Proof Size

In this subsection we only consider the proof size $s(\cdot)$ that counts the number of claims plus the length of all resolution refutations. If we use MICE' on unsatisfiable formulas, we have to prove that the formula has zero models. Hence, we can use MICE' as proof system for the language UNSAT as well. We show that MICE' is precisely as strong as resolution for unsatisfiable formulas.

▶ **Theorem 18.** MICE' *is polynomially equivalent to* Res *for unsatisfiable formulas.*

Proof Sketch. Let φ be an arbitrary unsatisfiable formula.

We first show that Res is simulated by MICE'. Suppose π_{Res} is a resolution refutation of φ, then we can use π_{Res} as an absence of models statement and derive the final claim $(\varphi, \emptyset, 0)$ with a single application of (Comp) of zero claims.

Next, we show that MICE' is simulated by Res. Let a MICE' refutation $\pi = (I_1, \dots, I_m)$ for φ be given with $I_i = (F_i, A_i, c_i)$. Further, let $\pi_{\mathsf{Res}} = (\varphi, X_1, X_2, \dots, X_m)$ where X_i is a sequence of clauses defined as

$$X_i := \begin{cases} \text{empty sequence} & \text{if } c_i \neq 0 \\ (\overline{A}_i) & \text{if } I_i \text{ is derived by (}\mathsf{Join'}\text{) or (}\mathsf{Ext'}\text{)} \\ (C \cup \overline{A}_i \mid C \in \rho) & \text{if } I_i \text{ is derived by (}\mathsf{Comp'}\text{) and absence of models statement } \rho. \end{cases}$$

We can show that π_{Res} is a valid resolution trace (with weakening steps). ◀

Many hard families of formulas for resolution are known. One famous example is the pigeonhole formula family PHP for which an exponential lower bound for resolution was first shown in [21]. With Theorem 18 we can conclude that these hard formulas for resolution are also hard for MICE'.

▶ **Corollary 19.** *Any* MICE' *proof* π *of* PHP_n *has size* $s(\pi) = 2^{\Omega(n)}$.

We note that it is also quite straightforward to obtain exponential proof size lower bounds for satisfiable formulas in MICE' by forcing the system to refute some exponentially hard CNFs in absence of models statements.

5.2 Lower Bounds for the Number of Inference Steps

One could argue that unsatisfiable formulas such as PHP are not particularly interesting for model counting. We also note that they have very simple MICE' proofs of just one step (as in the simulation of resolution by MICE' in Theorem 18) and that their hardness for MICE'

stems solely from the fact that they are hard for resolution (and such resolution proofs need to be included as an absence of models statement). However, we would argue that this does not tell us much on the complexity of MICE' proofs.

We therefore now tighten our complexity measure and consider the proof size measure $c(\cdot)$ that only counts the number of MICE' inference steps which is exactly the number of claims a proof π has. This measure disregards the size of the accompanying resolution refutations and hence formulas such as PHP become easy.

In our main result we present a family of formulas that is exponentially hard with respect to this sharper measure of counting inference steps. Such hard formulas need to have many models as the following upper bound shows.

▶ **Observation 20.** *Every formula φ has a MICE' proof π with $c(\pi) = |\mathsf{Mod}(\varphi)| + 2$.*

Proof. The MICE' proof that we used to show the completeness in Theorem 14 needs one (Ax) step, $|\mathsf{Mod}(\varphi)|$ applications of (Ext'), and one application of (Comp'). ◀

Therefore, to show exponential lower bounds to the number of steps we will need formulas with $2^{\Omega(n)}$ models. Next, we show that MICE' proofs for such formulas do not require claims with $c = 0$. In particular, we can assume that there are no such claims in the proofs.

▶ **Lemma 21.** *Let $\varphi \in SAT$ and π be a MICE' proof of φ. Then there is a MICE' proof π' of φ that has no claim with count $c = 0$ such that $s(\pi') = O(s(\pi)^2)$ and $c(\pi') \leq c(\pi)$.*

Proof Sketch. We consider an arbitrary claim I in the π with $c = 0$. Since I is not redundant, there is a path to the final claim. The final claim has count $c > 0$, since φ is satisfiable. Therefore, in this path there are two adjacent claims (F_1, A_1, c_1) and (F_2, A_2, c_2) with $c_1 = 0$ and $c_2 > 0$. We can argue that (F_2, A_2, c_2) is derived with (Comp'). We can adapt the absence of models statement such that (F_1, A_1, c_1) is not needed for this (Comp') application. ◀

Next, we introduce the family of formulas $(\mathsf{XOR\text{-}PAIRS}_n)_{n\in\mathbb{N}}$. They consist of variables x_i and z_{ij} for $i, j \in [n]$ and are satisfied exactly if $(z_{ij} = x_i \oplus x_j)$ for every pair $i, j \in [n]$.

▶ **Definition 22.** *The formula $\mathsf{XOR\text{-}PAIRS}_n$ consists of the clauses*

$$C_{ij}^1 = (x_i \vee x_j \vee \overline{z}_{ij}),\ C_{ij}^2 = (\overline{x}_i \vee x_j \vee z_{ij}),\ C_{ij}^3 = (x_i \vee \overline{x}_j \vee z_{ij}),\ C_{ij}^4 = (\overline{x}_i \vee \overline{x}_j \vee \overline{z}_{ij})$$

for $i, j \in [n]$.

▶ **Theorem 23.** *Any MICE' proof π of $\mathsf{XOR\text{-}PAIRS}_n$ requires size $c(\pi) = 2^{\Omega(n)}$.*

We start with some observations and lemmas and prove the lower bound at the end of this section.

The *idea of the proof* is the following: The final claim has a large count. In order to get a large count with a small number of MICE' steps, we have to use (Ext') or (Join') such that the previous counts get multiplied. However, we show that one factor of any such multiplication is always 1. As a result, the only way to increase the count is with (Comp'). We start with applications of (Ax) with count 1 and can only sum up those counts with (Comp'). As a result, we need an exponential number of summands.

▶ **Observation 24.** $\mathsf{XOR\text{-}PAIRS}_n$ *has 2^n models.*

Proof. We can set x_i arbitrarily for all $i \in [n]$ and have a unique assignment for the remaining z variables to satisfy $\mathsf{XOR\text{-}PAIRS}_n$. ◀

For the following arguments we will only consider MICE' proofs of $\mathsf{XOR\text{-}PAIRS}_n$ without redundant claims (i.e. all claims are connected to the final claim) and without claims with $c = 0$ (this is possible by Lemma 21). Our next lemma states that if we have some clause C_{ij} in a claim, then all missing clauses C_{ij} have to be satisfied by the assumption.

▶ **Lemma 25.** *Let (F, A) be an arbitrary claim in a MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$. If there are $i, j \in [n]$ such that $\{x_i, x_j, z_{ij}\} \subseteq \mathsf{vars}(F)$, then A has to satisfy every clause C^k_{ij} for $k \in [4]$ that is not in F.*

Proof Sketch. We fix variables $i, j \in [n]$ such that $\{x_i, x_j, z_{ij}\} \subseteq \mathsf{vars}(F)$ and a clause $C = C^k_{ij} \notin F$ for some $k \in [4]$. We consider only the path from (F, A) to $(\mathsf{XOR\text{-}PAIRS}_n, \emptyset)$ which has to exist, because otherwise (F, A) is redundant. There have to be claims $I_1 = (F_1, A_1)$ and $I_2 = (F_2, A_2)$ directly adjacent in this path with $F \subseteq F_1 \subseteq F_2 \subseteq \varphi$, $C \notin F_1$, $C \in F_2$, i.e. I_1 is the last claim in the path that does not contain C. I_2 is directly derived from I_1 with one of the four MICE' steps. We can argue that this is only possible if A satisfies C. ◀

The following lemma is similar in spirit. It shows that if all clauses C_{ij} are missing in a claim, then x_i and x_j have to be set in the assumption.

▶ **Lemma 26.** *Let a MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$ be given and let (F, A) be an arbitrary claim in the proof. If there are $i, j \in [n]$ such that $\{x_i, x_j\} \subseteq \mathsf{vars}(F)$ and $z_{ij} \notin \mathsf{vars}(F)$, then $\{x_i, x_j\} \subseteq \mathsf{vars}(A)$.*

Proof Sketch. The proof is very similar to the one of Lemma 25. We consider a path from (F, A) to the final claim and have a closer look at the first claim in this path that contains a clause C^k_{ij} for some $k \in [4]$. We argue that we can only derive this claim if $\{x_i, x_j\} \subseteq \mathsf{vars}(A)$ is fulfilled. ◀

Using the previous two lemmas, we show that the two inference rules that multiply counts, i.e. (Join') and (Ext'), do not affect the count at all for the $\mathsf{XOR\text{-}PAIRS}$ formulas.

▶ **Lemma 27.** *Let a MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$ be given. If the proof contains a (Join') of two claims (F_1, A_1, c_1) and (F_2, A_2, c_2), then $\min(c_1, c_2) = 1$.*

Proof. Suppose otherwise, $c_1 \geq 2$ and $c_2 \geq 2$.

Assume that all x variables occurring in $\mathsf{vars}(F_1)$ are assigned in A_1. Since $c_1 \geq 2$, $\mathsf{vars}(F_1) \setminus \mathsf{vars}(A_1) \neq \emptyset$. In particular, there has to be a $z_{ij} \in \mathsf{vars}(F_1) \setminus \mathsf{vars}(A_1)$ such that there is at least one model of F_1 and A_1 with $z_{ij} = 0$ and one with $z_{ij} = 1$. Then we have $\{x_i, x_j\} \subseteq \mathsf{vars}(F_1)$ and $\{x_i, x_j\} \subseteq \mathsf{vars}(A_1)$. As a result, A_1 has to satisfy all clauses C^k_{ij} that are in F_1. Because of Lemma 25, A_1 has to satisfy the clauses C^k_{ij} that are not in F_1 as well. Thus, A_1 has to satisfy all four clauses C^k_{ij}, which is only possible if $z_{ij} \in \mathsf{vars}(A_1)$. This contradicts the choice of z_{ij}. Similarly, we also see that there is at least one x variable in $\mathsf{vars}(F_2) \setminus \mathsf{vars}(A_2)$.

Hence, we can fix $x_i \in \mathsf{vars}(F_1) \setminus \mathsf{vars}(A_1)$ and $x_j \in \mathsf{vars}(F_2) \setminus \mathsf{vars}(A_2)$. Condition (J-2) implies $x_i \notin \mathsf{vars}(F_2)$, $x_j \notin \mathsf{vars}(F_1)$ and in particular $i \neq j$. Because of $\mathsf{vars}(A_1) \subseteq \mathsf{vars}(F_1)$ and $x_j \notin \mathsf{vars}(F_1)$ we get $x_j \notin \mathsf{vars}(A_1)$ and therefore also $x_j \notin \mathsf{vars}(A_1 \cup A_2)$. The joined claim is $(F, A) = (F_1 \cup F_2, A_1 \cup A_2)$ with $\{x_i, x_j\} \subseteq \mathsf{vars}(F)$ and $C^k_{ij} \notin F$ for all k, implying $z_{ij} \notin \mathsf{vars}(F)$. With Lemma 26 we get the contradiction $x_j \in \mathsf{vars}(A) = \mathsf{vars}(A_1 \cup A_2)$.

Therefore, our assumption $c_1 \geq 2$ and $c_2 \geq 2$ was false. ◀

Using this lemma we can show, that w.l.o.g. any MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$ does not use (Join').

▶ **Lemma 28.** *Let π be a MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$. Then there is a MICE' proof π' that does not use (Join') with $c(\pi') \leq 2 \cdot c(\pi)$.*

Proof. Using π we construct a MICE' proof π' that does not use (Join').

For this suppose that in π, the claim $I = (F_1 \cup F_2, A_1 \cup A_2)$ is derived with (Join') of (F_1, A_1, c_1) and (F_2, A_2, c_2). Because of Lemma 27 we can assume that $c_2 = 1$. Thus, there is a unique assignment α such that $\mathsf{vars}(A_2) \cap \mathsf{vars}(\alpha) = \emptyset$, $\mathsf{vars}(A_2 \cup \alpha) = \mathsf{vars}(F_2)$ and $A_2 \cup \alpha$ satisfies F_2. Then, we can apply (Ext') to (F_1, A_1) resulting in $(F_1 \cup F_2, A_1 \cup A_2 \cup \alpha)$. We check the conditions to apply (Ext').

(E-1) $F_1 \subseteq F_1 \cup F_2$ holds.

(E-2) We see that $(A_1 \cup A_2 \cup \alpha)|_{\mathsf{vars}(F_1)} = A_1|_{\mathsf{vars}(F_1)} \cup A_2|_{\mathsf{vars}(F_1)} \cup \alpha|_{\mathsf{vars}(F_1)} = A_1$. In the last equation we used three facts:

$A_1|_{\mathsf{vars}(F_1)} = A_1$ is a direct consequence of $\mathsf{vars}(A_1) \subseteq \mathsf{vars}(F_1)$.

$A_2|_{\mathsf{vars}(F_1)} \subseteq A_1$ follows from $\mathsf{vars}(A_2|_{\mathsf{vars}(F_1)}) \subseteq \mathsf{vars}(F_2) \cap \mathsf{vars}(F_1) \subseteq \mathsf{vars}(A_1)$ by (J-2) and the fact that A_1 and A_2 are consistent by (J-1).

$\alpha|_{\mathsf{vars}(F_1)} = \emptyset$. Assume otherwise that $x \in \mathsf{vars}(\alpha) \cap \mathsf{vars}(F_1)$. Then $x \in \mathsf{vars}(\alpha) \cap \mathsf{vars}(F_1) \subseteq \mathsf{vars}(F_2) \cap \mathsf{vars}(F_1) \subseteq \mathsf{vars}(A_2)$ by (J-2). Thus, $x \in \mathsf{vars}(A_2) \cap \mathsf{vars}(\alpha)$ contradicting the construction of α.

(E-3) $A_1 \cup A_2 \cup \alpha$ satisfies $(F_1 \cup F_2) \setminus F_1 \subseteq F_2$ as $A_2 \cup \alpha$ satisfies F_2 by construction.

Applying (Comp') on the claim $(F_1 \cup F_2, A_1 \cup A_2 \cup \alpha)$ we get $(F_1 \cup F_2, A_1 \cup A_2)$. In this way we can remove every (Join') application with one application of each (Ext') and (Comp'). Let π' be the resulting MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$ that does not use (Join'). The number of claims in the proof increases at most by a factor of two. ◀

▶ **Lemma 29.** *Let a MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$ be given. Any claim (F, A, c) in the proof that is derived with (Ext') from (F_1, A_1, c_1) satisfies $c = c_1$.*

Proof. Suppose $c \neq c_1$. Since $c = c_1 \cdot 2^{|\mathsf{vars}(F) \setminus (\mathsf{vars}(F_1) \cup \mathsf{vars}(A))|}$ there is a variable $v \in \mathsf{vars}(F)$ with $v \notin \mathsf{vars}(F_1) \cup \mathsf{vars}(A)$. Variable v occurs in some clause $C_{ij}^k \in F \setminus F_1$. Therefore, $\{x_i, x_j, z_{ij}\} \subseteq \mathsf{vars}(F)$. A has to satisfy all clauses of C_{ij} that occur in $F \setminus F_1$ because of (E-3). Furthermore, A has to satisfy all clauses of C_{ij} that do not occur in F as well due to Lemma 25. Since, $v \notin \mathsf{vars}(F_1)$, there is no $C_{ij} \in F_1$. Therefore, A has to satisfy all four clauses C_{ij}. For this, x_i, x_j and z_{ij} have to be set in A. Since v occurs in C_{ij}, we have $v \in \mathsf{vars}(A)$ which contradicts the choice of v. ◀

Now we have all ingredients to finally prove that the XOR-PAIRS formulas require proofs with an exponential number of MICE' steps.

Proof of Theorem 23. Note that with Observation 24, Lemma 27 and Lemma 29 we can infer immediately that any tree-like MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$, i.e. any proof that uses every claim except axiom at most one time, has at least size $2^n + 2$. However, dag-like MICE' might be stronger than tree-like MICE'. Therefore, the lower bound is not shown yet.

To prove the lower bound in the general case, let π be an arbitrary MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$. Let π' be a MICE' proof of $\mathsf{XOR\text{-}PAIRS}_n$ that does not use (Join') with $c(\pi') \leq 2 \cdot c(\pi)$ which has to exist because of Lemma 28.

We consider an arbitrary fixed path κ in π' from the axiom to the final claim. Since π' does not use (Join'), we can only enlarge the formula with (Ext'). Because of Lemma 29, we have to assign all newly introduced variables when we use (Ext'), i.e. every variable is in at least one assumption in κ. The only rule that can remove variables from the assumption is (Comp').

Since the final claim has the empty assumption, we have to remove all variables from the assumption in κ. Therefore, in κ there has to be at least one application of (Comp') where we remove a variable x_i from the assumption for some $i \in [n]$. Let $I_1^\kappa = (F_1^\kappa, A_1^\kappa)$ be the claim that was used for the first such (Comp') in κ to derive $I_2^\kappa = (F_2^\kappa, A_2^\kappa)$.

Let X be the set of all x variables: $X := \{x_1, \ldots, x_n\}$. We show

$$X \subseteq \mathsf{vars}(F_1^\kappa).$$

Let x_i be a variable that is removed from the assumption by applying (Comp') to I_1^κ, i.e. $x_i \notin \mathsf{vars}(A_2^\kappa)$. Suppose, there is a $j \in [n]$ such that $x_j \notin \mathsf{vars}(F_1^\kappa)$ and in particular $C_{ij}^s \notin F_1^\kappa$ for all $s \in [4]$, implying $z_{ij} \notin \mathsf{vars}(F_1^\kappa)$. Let $I_r^\kappa = (F_r^\kappa, A_r^\kappa)$ be the first claim in κ with $z_{ij} \in \mathsf{vars}(F_r^\kappa)$ and therefore $\{x_i, x_j, z_{ij}\} \subseteq \mathsf{vars}(F_r^\kappa)$. I_r^κ has to be derived with (Ext'). Because of condition (E-3), A_r^κ has to satisfy all clauses C_{ij}^s in F_r^κ. Furthermore, A_r^κ has to satisfy all clauses C_{ij}^s that are not in F_r^κ because of Lemma 25. Hence, A_r^κ has to satisfy C_{ij}^s for all $s \in [4]$. To do so, we have to assign all three variables x_i, x_j and z_{ij} in A_r^κ. In particular, we have $x_i \in \mathsf{vars}(A_r^\kappa)$. Since $x_i \notin \mathsf{vars}(A_2^\kappa)$, Lemma 15 states $x_i \notin \mathsf{vars}(A_r^\kappa)$. With this contradiction we see that such an x_j with $x_j \notin \mathsf{vars}(F_1^\kappa)$ cannot exist.

Since $X \subseteq \mathsf{vars}(F_1^\kappa)$, all variables in X were introduced and assigned in the assumption with (Ext') in I_1^κ or previously in κ. Per construction there are no other (Comp') applications before I_1^κ in κ that remove variables in X. Therefore, we have

$$X \subseteq \mathsf{vars}(A_1^\kappa).$$

We show that for every $\alpha \in \mathsf{Mod}(\mathsf{XOR\text{-}PAIRS}_n)$ there is a path κ in π' with $\alpha|_X = A_1^\kappa|_X$. Assume that for some fixed model α there is no such path. Since π' does not use (Join') and $\alpha \in \mathsf{Mod}_\emptyset(\mathsf{XOR\text{-}PAIRS}_n)$, Corollary 11 implies that there is a path κ from axiom to the final claim, such that every claim (F, A) in κ fulfills $\alpha|_{\mathsf{vars}(F)} \in \mathsf{Mod}_A(F)$. In particular,

$$\alpha|_{\mathsf{vars}(F_1^\kappa)} \in \mathsf{Mod}_{A_1^\kappa}(F_1^\kappa).$$

If we restrict both sides on the variables in X and use $X \subseteq \mathsf{vars}(F_1^\kappa)$, we get

$$\alpha|_X \in \{\beta|_X \mid \beta \in \mathsf{Mod}_{A_1^\kappa}(F_1^\kappa)\}.$$

Since $X \subseteq \mathsf{vars}(A_1^\kappa)$, all models $\beta \in \mathsf{Mod}_{A_1^\kappa}(F_1^\kappa)$ have $\beta|_X = (A_1^\kappa)|_X$. Therefore, the right set has only one element which is $(A_1^\kappa)|_X$, leading to $\alpha|_X = (A_1^\kappa)|_X$. Hence, κ is a path with the claimed property for α.

Since $\mathsf{XOR\text{-}PAIRS}_n$ has 2^n models, there are (at least) 2^n paths in π' and in particular 2^n claims I_1^κ. Because every model of $\mathsf{XOR\text{-}PAIRS}_n$ assigns the x variables differently, all these claims I_1^κ are pairwise different. Therefore, π' has at least 2^n claims.

Finally, we see that the arbitrarily chosen MICE' proof π has size $c(\pi) \geq \frac{1}{2} \cdot c(\pi') \geq 2^{n-1}$ leading to the lower bound. ◀

6 Conclusion

We performed a proof-complexity study of the #SAT proof system MICE, exhibiting hard formulas, both in terms of unsatisfiable CNFs, where their complexity in MICE matches their resolution complexity, and for highly satisfiable CNFs with many models. As Fichte et al. [18] show that MICE proofs can be extracted from solver runs for sharpSAT [29], DPDB [19] and D4 [24], this implies a number of hard instances for these #SAT solvers.

We believe that the ideas for the lower bound for our formula XOR-PAIRS can be extended to show hardness of further CNFs with many models. A natural problem for future research is to construct stronger #SAT proof systems (and #SAT solvers) where formulas such as XOR-PAIRS become easy.

As pointed out by one reviewer and mentioned in Section 1.2, there appears to be a close connection between MICE′ proofs and decision DNNFs. Therefore, it seems promising to investigate if known results from decision DNNFs can be transferred to MICE′. This may lead to more hard formulas and lower bounds for MICE′.

It would also be interesting to determine the exact relations between the systems MICE, MICE′ and the kcps(#SAT) proof system from [11] based on certified decision DNNFs.

References

1 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011.

2 Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #sat and bayesian inference. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 340–351. IEEE Computer Society, 2003.

3 Teodora Baluta, Zheng Leong Chua, Kuldeep S. Meel, and Prateek Saxena. Scalable quantitative verification for deep neural networks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 312–323. IEEE, 2021.

4 Olaf Beyersdorff and Benjamin Böhm. Understanding the Relative Strength of QBF CDCL Solvers and QBF Resolution. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:20, 2021.

5 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.

6 Benjamin Böhm and Olaf Beyersdorff. Lower bounds for QCDCL via formula gauge. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing (SAT)*, pages 47–63, Cham, 2021. Springer International Publishing.

7 Benjamin Böhm, Tomás Peitl, and Olaf Beyersdorff. QCDCL with cube learning or pure literal elimination – What is best? In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1781–1787. ijcai.org, 2022.

8 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Knowledge compilation meets communication complexity. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1008–1014. IJCAI/AAAI Press, 2016.

9 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, pages 233–350. IOS Press, 2021.

10 Sam Buss and Neil Thapen. DRAT proofs, propagation redundancy, and extended resolution. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing (SAT)*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2019.

11 Florent Capelli. Knowledge compilation languages as proof systems. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing (SAT)*, volume 11628 of *Lecture Notes in Computer Science*, pages 90–99. Springer, 2019.

12 Florent Capelli, Jean-Marie Lagniez, and Pierre Marquis. Certifying top-down decision-dnnf compilers. In *Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI) 2021*, pages 6244–6253. AAAI Press, 2021.

13 Leroy Chew and Marijn J. H. Heule. Relating existing powerful proof systems for QBF. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 236 of *LIPIcs*, pages 10:1–10:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

14 Stephen A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

15 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symb. Log.*, 44(1):36–50, 1979.

16 Leonardo Dueñas-Osorio, Kuldeep S. Meel, Roger Paredes, and Moshe Y. Vardi. Counting-based reliability estimation for power-transmission grids. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4488–4494. AAAI Press, 2017.

17 Johannes Klaus Fichte, Markus Hecher, and Florim Hamiti. The model counting competition 2020. *ACM J. Exp. Algorithmics*, 26:13:1–13:26, 2021.

18 Johannes Klaus Fichte, Markus Hecher, and Valentin Roland. Proofs for propositional model counting. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 30:1–30:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

19 Johannes Klaus Fichte, Markus Hecher, Patrick Thier, and Stefan Woltran. Exploiting database management systems and treewidth for counting. In Ekaterina Komendantskaya and Yanhong Annie Liu, editors, *Practical Aspects of Declarative Languages – 22nd International Symposium, PADL 2020, New Orleans, LA, USA, January 20-21, 2020, Proceedings*, volume 12007 of *Lecture Notes in Computer Science*, pages 151–167. Springer, 2020.

20 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 993–1014. IOS Press, 2021.

21 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

22 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Automated Deduction – CADE-24 – 24th International Conference on Automated Deduction*, pages 345–359, 2013.

23 Marijn J. H. Heule, Martina Seidl, and Armin Biere. Solution validation and extraction for QBF preprocessing. *J. Autom. Reason.*, 58(1):97–125, 2017.

24 Jean-Marie Lagniez and Pierre Marquis. An improved decision-dnnf compiler. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 667–673. ijcai.org, 2017.

25 Anna L. D. Latour, Behrouz Babaki, Anton Dries, Angelika Kimmig, Guy Van den Broeck, and Siegfried Nijssen. Combining stochastic constraint optimization and probabilistic programming – From knowledge compilation to constraint solving. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming – 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 495–511. Springer, 2017.

26 João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.

27 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. `doi:10.1016/j.artint.2010.10.002`.

28 Weijia Shi, Andy Shih, Adnan Darwiche, and Arthur Choi. On tractable representations of binary neural networks. In Diego Calvanese, Esra Erdem, and Michael Thielscher, editors, *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, pages 882–892, 2020.

29 Marc Thurley. sharpSAT – Counting models with advanced component caching and implicit BCP. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing – SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 424–429. Springer, 2006.

30 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

31 Moshe Y. Vardi. Boolean satisfiability: Theory and engineering. *Commun. ACM*, 57(3):5, 2014.

32 Marc Vinyals. Hard examples for common variable decision heuristics. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

33 Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing (SAT*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014.

34 Ennan Zhai, Ang Chen, Ruzica Piskac, Mahesh Balakrishnan, Bingchuan Tian, Bo Song, and Haoliang Zhang. Check before you change: Preventing correlated failures in service updates. In Ranjita Bhagwan and George Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 575–589. USENIX Association, 2020.

CadiBack: Extracting Backbones with CaDiCaL

Armin Biere
Universität Freiburg, Germany

Nils Froleyks
Johannes Kepler Universität, Linz, Austria

Wenxi Wang
University of Texas at Austin, TX, USA

Abstract

The backbone of a satisfiable formula is the set of literals that are true in all its satisfying assignments. Backbone computation can improve a wide range of SAT-based applications, such as verification, fault localization and product configuration. In this tool paper, we introduce a new backbone extraction tool called CadiBack. It takes advantage of unique features available in our state-of-the-art SAT solver CaDiCaL including transparent inprocessing and single clause assumptions, which have not been evaluated in this context before. In addition, CaDiCaL is enhanced with an improved algorithm to support model rotation by utilizing watched literal data structures. In our comprehensive experiments with a large number of benchmarks, CadiBack solves 60% more instances than the state-of-the-art backbone extraction tool MiniBones. Our tool is thoroughly tested with fuzzing, internal correctness checking and cross-checking on a large benchmark set. It is publicly available as open source, well documented and easy to extend.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases Satisfiability, Backbone, Incremental Solving

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.3

Supplementary Material *Software*: `https://github.com/arminbiere/cadiback`
archived at `swh:1:dir:9e9c1ac209792174194cde74eb3a385f48ceade2`

1 Introduction

In 1997, Parkes first defined the backbone of a propositional formula as the set of literals whose assignments are true in every satisfying assignment [24]. The size of the backbone is associated with the hardness of the corresponding propositional problem [23, 27]. Usually, the larger a backbone, the more tightly constrained the problem becomes, thus the harder for the solver to find a satisfying assignment [11, 30]. It is proved by Janota that deciding if a literal is in the backbone of a formula is co-NP complete [17]. Furthermore, Kilby et al. show that even approximating the backbone is intractable in general [20].

Nevertheless, the identification of the backbone (either in a partial or a complete way) has a number of practical applications, such as post-silicon fault localization in integrated circuits [34, 36, 35], interactive product configuration [17], facilitating the solving efficiency of MaxSAT [14, 28, 29, 31] and random 3-SAT problems [12], as well as improving the performance of chip verification [26]. Motivated by the wide range of applications, developing efficient algorithms for computing the backbone of a given propositional formula is important.

Indeed, numerous techniques to compute the backbone have been proposed during the past few decades. These approaches make use of four main techniques: (*i*) model enumeration, which enumerates all models of a satisfiable formula to identify the backbone; (*ii*) iterative SAT testing, which repeatedly filters out a candidate or include it in the backbone; (*iii*) upper bound checks, which try to identify multiple backbone literals at once; and (*iv*) the core-based method, which is guided by unsatisfiable cores and tries to eliminate as many candidates at once as possible. For example, Kaiser et al. [19] designed three model-enumeration algorithms. Climer et al. [10] propose a graph-based iterative SAT testing approach.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 3; pp. 3:1–3:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Later, Zhu et al. [34, 36] designed more efficient SAT testing approaches for post-silicon fault localization. Note that, the backbone extractor MiniBones [18, 22] implements both an iterative and a core-based approach. Despite recent attempts [25, 32, 33] to improve upon MiniBones, the corresponding tools are not publicly available, and no significant advances have been made so far which still leaves MiniBones as the state-of-the-art.

Our new backbone extractor CadiBack tries to improve the iterative algorithms of MiniBones [18, 22] and uses the state-of-the-art SAT solver CaDiCaL [4], extended with new flipping algorithms to support backbone extraction. Different configurations of these algorithms are implemented inside CadiBack, empirically evaluated and compared with MiniBones on a large set of satisfiable instances collected from the main track of the SAT Competitions from 2004 - 2022, on which CadiBack solves 60% more instances.

The paper is structured as follows. After this introduction, we discuss basic concepts and notations related to backbone extraction in Section 2. The relevant backbone extraction algorithms of MiniBones are introduced in Section 3. We then present our improvements over these algorithms and propose CadiBack in Section 4. The implementation details of CadiBack are provided in Section 5. Finally, we empirically evaluate CadiBack in Section 6 and draw conclusions in Section 7.

2 Basic Concepts and Notations

Consider a propositional *formula* φ in conjunctive normal form (CNF) over a fixed set of variables $\mathcal{V}$ and literals $\mathcal{L} = \mathcal{V} \cup \overline{\mathcal{V}}$, where $\overline{\mathcal{V}} = \{\bar{v} \mid v \in \mathcal{V}\}$ denotes the negated variables. For a literal $\ell \in \mathcal{L}$, we define $v = |\ell|$ as the variable of ℓ, i.e., $\ell \in \{v, \bar{v}\}$. In this paper, we mainly consider full *assignments* $\sigma \colon \mathcal{V} \rightarrow \{0, 1\}$ assigning variables to Boolean constants "0" (*false*) or "1" (*true*). For convenience, we use the set and logic notation interchangeably for formulas φ, clauses $C \in \varphi$ and literals $\ell \in C$, as well as assignments $\{\ell \mid \sigma(\ell) = 1\}$. The notion of assignments is lifted to literals, formulas and clauses in the natural way through substitution followed by Boolean expression simplification. A *model* of φ is an assignment σ with $\sigma(\varphi) = 1$ and also called satisfying assignment. A formula is *satisfiable* if it has a model. Otherwise, it is *unsatisfiable*. In this paper, we focus on satisfiable formulas φ.

A literal ℓ is a *backbone literal* of a formula φ iff there exists a model σ of φ with $\sigma(\ell) = 1$ and all other assignments σ' with $\sigma'(\ell) = 0$ do not satisfy φ, i.e., $\sigma'(\varphi) = 0$. The *backbone* $\mathcal{B}$ of a formula φ is the set of its backbone literals. We introduce two conditions that determine whether literals are included or not in the backbone $\mathcal{B}$.

The first condition is based on identifying *fixed* literals. A clause $C = \ell$ (or $C = \{\ell\}$ in set notation) having a single literal ℓ is called *unit* clause. If a unit clause $C \in \varphi$, the corresponding literal ℓ is clearly a backbone literal. We call such literals *fixed*. This also applies to unit clauses deduced by the SAT solver through for instance clause learning, simplification and preprocessing [7]. All such fixed literals are included in the backbone $\mathcal{B}$.

The second condition is called *disagreement condition*, stating that if there are two models σ and σ' *disagreeing* on ℓ, i.e., $\sigma'(\ell) = \overline{\sigma(\ell)}$, then neither ℓ nor its negation are backbone literals (i.e., $\ell, \overline{\ell} \notin \mathcal{B}$). This can be realized by using each newly discovered model σ' to *filter* the list of remaining backbone candidates. For instance, the empty formula over n variables has both constant assignments $\sigma \equiv 0$ and $\sigma' \equiv 1$ as models, disagreeing on all literals, and thus $\mathcal{B} = \emptyset$. Note that, there is a special case of the disagreement condition called *model rotation*, as described in [18]. Similar ideas have been used for MUS extraction [1]. The literal ℓ is *rotatable* [18] in a model σ of φ iff $\sigma(\ell) = 1$ and the assignment τ that differs from σ only in $|\ell|$ is a model of φ (τ can be taken as the special case of σ' in the disagreement condition). We also call such literals *flippable*, which applies for the rest of the paper.

Obviously, a literal which can be flipped is not a backbone literal, nor is its negation, and both can be dropped from the backbone candidate list. Example 1 below shows how a literal is determined to be flippable under the model rotation condition.

▶ **Example 1.** Consider $\varphi = (\bar{c} \vee t) \wedge (c \vee e) \wedge \varphi'$ which encodes "if-then-else(c, t, e)", where neither c nor $\bar{c}$ occur in φ' but e and t do (they are not "pure"). Assume that the constant true assignment $\sigma \equiv 1$ is a model, i.e., $\sigma(\varphi') = 1$. Both t and e are set to true, but only the literal c can be flipped. In the resulting model τ, all variables are set to true except for c, and $\bar{c}$ can be flipped (back) in τ to obtain the original model σ. Thus, literal c is flippable.

3 Algorithms in MiniBones

The backbone extraction algorithms of MiniBones [18] take advantage of incremental SAT solving (refer to [13, 15, 16] for details) to gradually augment the original formula with implied clauses (particularly learned clauses). These clauses are added implicitly to the single SAT solver instance during incremental queries, while assuming the negation of one or more remaining backbone candidate literals. Specifically, these iterative MiniBones algorithms (Algorithms 3, 4 and 5 in [18]) utilize discovered models and model rotation to refine the set of candidate literals $\Lambda \subseteq \mathcal{L}$ which is initialized as $\Lambda = \{\ell \mid \sigma(\ell) = 1\}$ by the first discovered model σ. On termination ($\Lambda = \emptyset$), the backbone $\mathcal{B}$ matches the fixed literals (of the augmented formula) and all other literals are dropped.

There are three iterative algorithms proposed for MiniBones in [18]. The basic algorithm (Algorithm 3 in [18]) needs at least as many iterations as the number of backbone literals, which is inefficient on formulas with exactly one solution but many variables. An improved algorithm (Algorithm 4 in [18]) assumes that at least one of the remaining candidate literals can be flipped (i.e., using activation literals a temporary clause is added that contains the disjunction of the negated candidates). If the SAT query under such assumption is unsatisfiable, all candidates are fixed and the backbone extraction is done. A more advanced algorithm (Algorithm 5 in [18]) only adds a subset of the remaining candidates, called a *chunk*, to the temporary clause. Chunks are limited in size to avoid thrashing the SAT solver with too large temporary clauses and make it more likely for a call to be unsatisfiable.

Furthermore, MiniBones proposes a new model rotation algorithm (Section 5 in [18]) to determine flippable (rotatable) literals based on the notion of *forcing*. A clause C *forces* a literal $\ell \in C$ under assignment σ, if $\sigma(C) = \sigma(\ell) = 1$ and $\tau(C) = 0$ with τ obtained from σ by flipping ℓ. A literal ℓ is *forced* in a formula φ under a model σ, if there is a clause $C \in \varphi$ which forces ℓ under σ. It is straightforward to see that literals which can be flipped in a model σ of φ are exactly those that are not forced. Based on this observation, the model rotation algorithm goes over all clauses whenever a new model is found and identifies literals that are not forced by any of them. If any of the remaining backbone candidates are not forced, they are dropped from the candidate list.

4 Improved Algorithms in CadiBack

CadiBack is built upon the state-of-the-art SAT solver CaDiCaL [4] which has been extended with additional algorithms to support backbone extraction. The general backbone extraction algorithm of CadiBack is shown in Algorithm 1 of Figure 1. It follows the iterative algorithms of MiniBones, which uses complements of backbone estimates (as constraints) and chunking, but with three key improvements.

```
// Assume $\varphi$ is satisfiable and use
// $K = 1$ for one-by-one,
// $K = 10$ for chunking and
// $K = \infty$ as default (non-chunking).
backbone (CNF $\varphi$, chunk rate $K = \infty$)
  $(res, \sigma) \leftarrow$ SAT($\varphi$)
  assert $\sigma(\varphi) = res = 1$    // $\varphi$ satisfiable!
  $\Lambda \leftarrow \{\ell \in \sigma \mid \neg$flippable$(\ell, \sigma)\}$  // candidates
  $k \leftarrow 1$,   $\mathcal{B} \leftarrow \emptyset$
  while $\Lambda \neq \emptyset$ do
      // $F \leftarrow \emptyset$ for no-fixed, otherwise by default
      $F \leftarrow \{\ell \in \Lambda \mid \ell$ is fixed by SAT in $\varphi\}$
      $\mathcal{B} \leftarrow \mathcal{B} \cup F$,   $\Lambda \leftarrow \Lambda \setminus F$
      $\Gamma \leftarrow$ pick $k'$ literals from $\Lambda$     // chunk
            with $k' = \min(k, |\Lambda|)$
      $\rho \leftarrow \bigvee_{\ell \in \Gamma} \bar{\ell}$  // constraint: flip one in chunk
      // Solve $\varphi$ under $\rho$ with "bool constrain"
      // or use activation literal for no-constrain.
      $(res, \sigma) \leftarrow$ SAT($\varphi \mid \rho$)
      if $res$ then     // SAT call satisfiable
          // filter only a single literal for no-filter
          $\Lambda \leftarrow \{\ell \in \Lambda \mid \sigma(\ell)\}$
          $\Lambda \leftarrow \{\ell \in \Lambda \mid \neg$flippable$(\ell, \sigma)\}$
          $k \leftarrow 1$  // reset chunk size to 1
      else              // SAT call unsatisfiable
          $\mathcal{B} \leftarrow \mathcal{B} \cup \Gamma$
          $\Lambda \leftarrow \Lambda \setminus \Gamma$
          $k \leftarrow K \cdot k$  // increase size geometrically
  return $\mathcal{B}$    // or print when literal is added
```

Algorithm 1 Extracting backbone of formula φ.

```
// Assume $\sigma(\varphi) = \sigma(\ell) = 1$, unit clauses
// have exactly one watched literal
// and all other clauses are watched
// by two literals $w_1 \neq w_2$ with
// $\sigma(w_1) = 1$ if $\sigma(w_2) = 0$ and vice versa.
flippable (CNF $\varphi$, literal $\ell$, model $\sigma$)
  // return 0 for no-flip
  for all clauses $C$ watched by $\ell$ in $\varphi$
      if $\sigma(C \setminus \{\ell\}) = 0$ then return 0
  return 1
```

Algorithm 2 Checking if literal ℓ can be flipped in model σ.

```
// Given a single clausal constraint
// $\rho = \ell_1 \vee \cdots \vee \ell_k$ and assignment $\sigma$
// determine whether $\rho$ is conflicting.
// Otherwise pick new decision.
decide (constraint $\rho$, partial model $\sigma$)
  ... // handle literal assumptions
  if $\sigma(\rho) = 1$ then // constraint true
      $\ell \leftarrow$ "first" literal in $\rho$ with $\sigma(\ell)$
      // speed-up future search for $\ell$
      move $\ell$ to the front of $\rho$
  elif $\sigma(\rho) = 0$ then // constraint false
      ... // handle conflicting constraint
  else    // constraint undetermined
      $\ell \leftarrow$ highest scored literal in $\sigma(\rho)$
      pick $\ell$ as new decision and return
  ... // fall back to default decisions
```

Algorithm 3 Picking the next decision literal under clausal constraint ρ and the partial model σ.

Figure 1 Our backbone algorithm combines all three iterative approaches from [18]. It simulates the basic iterative Algorithm 3 in [18] for $K = 1$ and comes close to the improved Algorithm 4 in [18] for $K > |\Lambda|$ and the most advanced Algorithm 5 in [18] for other values of K. The difference between our algorithm and the latter two is that we use a dynamic chunk size that is reset to 1 after a satisfiable call and grows geometrically as long SAT queries remain unsatisfiable. In any case, it first identifies an initial model σ and initializes the set of candidates Λ after filtering out flippable literals F. The remaining candidates are examined in chunks Γ. If all of the literals in the chunk are backbones, the chunk size is increased. Otherwise, the solver returns a new model σ which is used to filter the candidate list, as it is guaranteed to disagree with the previous model in at least one of the literals in the current chunk by assuming the constraint ρ. After that, another model rotation is performed and the chunk size is reset to 1. Note that, instead of including explicit insertions of backbones, we can assume that the SAT solver does the insertion implicitly. Flippable literals are identified by the new flippable algorithm which only traverses clauses watched by the remaining backbone candidates. The decide algorithm is an optimized version of the decision procedure in our SAT solver for more efficiently handling large constraints as they arise in this application, which picks the literal with the highest variables scores (i.e., EVSIDS scores [2] or VMTF stamps [5]).

First, CadiBack uses transparent incremental inprocessing [15], as CaDiCaL is able to effectively and efficiently simplify the formula (e.g., using variable elimination) during incremental queries completely transparent to the user, while MiniBones does not support inprocessing due to the limitation of its base solver MiniSat [13].

Second, to assume the disjunction of the complements, CadiBack utilizes single clause assumptions through the "`void constrain (int lit)`" API call in CaDiCaL [16], instead of adding a clause with the complement literals and an activation literal [13], as in MiniBones. The reason is that these added clauses and variables by MiniBones may risk to clog the SAT solver, and handling constraints explicitly can have the benefit to give the SAT solver more control on selecting decisions. Since the assumed clausal constraint contains a high number of literals in this application ($|\mathcal{V}|$ initially), we extended the existing implementation of single clause assumptions in CaDiCaL slightly, as shown in Algorithm 3 in Figure 1. After each restart the SAT solver is forced to decide on a literal to satisfy the constraint. CadiBack chooses the one with the highest variable score (EVSIDS scores [2] or VMTF stamps [5]) among all unassigned literals in the constraint.

Third, while in earlier work model rotation only had negative effects on MiniBones [18], we show that CadiBack benefits from using model rotation to improve efficiency of backbone computation. The key of this improvement is our fast flipping algorithm implemented in CaDiCaL, accessible through the new API call "`bool flippable (int lit)"`. As described in Algorithm 2 in Figure 1, it uses watch lists to find individual "flippable" literals in models through propagation instead of going over the whole formula to find unit clauses. We also consider a variant of Algorithm 2 which eagerly flips flippable literals as they are found, with the goal to drop even more backbone candidates through flipping. The following example shows the possibility that flipping a flippable literal can yield additional flippable literals.

▶ **Example 2.** Continuing Example 1, assume that no clause in φ' forces literal t under $\sigma \equiv 1$, which is not the case for the first clause $(\bar{c} \vee t)$ in φ, as it forces t under σ. Thus, t cannot be flipped in σ. As c does not occur in φ', there is no clause forcing $\bar{c}$ under τ. In addition, the only other clause $(\bar{c} \vee t)$ with t is not forcing as it is satisfied by two literals. Thus, flipping c makes t flippable in τ (τ' obtained from τ by flipping t remains a model of φ). Therefore, neither c nor t are backbone literals.

To implement this idea we provide a new "`bool flip (int lit)`" API call in CaDiCaL which implements a variant of Algorithm 2 in Figure 1, inspired by propagation in SAT solvers. While for "`flippable`" we only need to check that there is another satisfied literal in all traversed clauses watched by the literal ℓ requested to be flipped, the "`flip`" implementation needs to unwatch ℓ in these clauses and watch that other satisfied literal instead (unless the second watched literal in the clause is also satisfied). If finding replacements is successful for all clauses watched by ℓ (or the other watched literal is satisfied), the value of ℓ is flipped. Otherwise, it remains unchanged and "`flip`" fails. Note that this variant of our flipping algorithm was previously implemented inside the sub-solver Kitten of Kissat to diversify models with the goal of speeding up the refinement process of SAT sweeping [3].

Algorithm 3 of [18] can be simulated precisely with our algorithm by setting $K = 1$. However, Algorithm 5 of [18], which uses a fixed chunk size limit can only be approximated by setting $K = 100$, as we change the chunk size k dynamically. Our adaptive scheme increases k geometrically with rate K as long as SAT queries remain unsatisfiable (which fixes all backbones in the chunk at once). If the SAT solver finds a model instead then the chunk size k is reset to one, i.e., the next constraint will only contain the negation of a single backbone candidate. With $K = \infty$ the SAT solver is either assuming the complement of a single or the disjunction of the negation of all remaining backbone candidates which is the setting of our algorithm closest to Algorithm 4 of [18] which does not limit chunk size at all.

5 Implementation

Our tool CadiBack uses the extended CaDiCaL [4] and is implemented in roughly 1200 lines of C++ code (counted after formatting with ClangFormat). The source code available at `https://github.com/arminbiere/cadiback` is concise and well-documented.

To check the correctness of algorithms and implementations, an internal backbone checker is implemented inside CadiBack. The checker can be enabled through the command line option "`--check`" and is simply a SAT solver instance of CaDiCaL, i.e., if checking is enabled, CadiBack obtains the checker instance as a copy of the main internal CaDiCaL solver through the "`copy`" API call provided by CaDiCaL.

First, it checks correctness of an identified backbone literal ℓ, by confirming that the input formula φ under the assumption $\neg\ell$ (negation of the backbone) is unsatisfiable. Second, it checks the correctness of dropping a literal ℓ from being a backbone candidate (removed from set Λ in Algorithm 1), by confirming that the input formula φ remains satisfiable under the assumption $\neg\ell$. Third, it checks whether the number of backbone literals found and the number of dropped literals sum up to the number of variables in the input formula.

Standard grammar-based black-box fuzz-testing was applied [9] with the backbone checking enabled on all 42 compatible pairs of options used in our experiments in Section 6. This pairwise combinatorial testing [21] through fuzzing was run for 50 hours in parallel using as many processes as configurations on a dual processor AMD EPYC 7343 machine (providing in total 64 virtual cores). In addition, sizes of the backbones of all our benchmarks (see Section 6) were sanity checked with the ones computed by 12 configurations of CadiBack and two configurations of MiniBones considered in our experiments.

In addition, for flipping information extraction, the library of CaDiCaL is extended to provide "`bool flippable (int)`" and "`bool flip (int)`" as discussed in the last section. The model based tester Mobical is also extended correspondingly for testing the new functionality. This extended version of CaDiCaL with improved constrain handling and flipping is also available at `https://github.com/arminbiere/cadical`.

6 Experiments

Benchmarks. To evaluate CadiBack empirically, we collected all benchmarks from the main track of the SAT competition 2004 to 2022 as our initial benchmark set. We noticed that benchmarks from one competition year often contain old benchmarks (sometimes arbitrarily renamed or commented by the competition organizers) from previous competition years. This caused our initial benchmark set to include several redundant benchmarks. To remove such duplicates, in a second step, we cleaned up each individual benchmark by removing comments using a simple DIMACS pretty printer, followed by identifying identical benchmarks through computing an MD5 checksum and removing redundant ones. Then we ran the state-of-the-art SAT solver Kissat 3.0.0 [3] with 5,000 second timeout on the no-duplicate benchmark set and selected benchmarks solved to be satisfiable. In total this yields 1798 benchmarks available at `https://cca.informatik.uni-freiburg.de/sc04to22sat.zip` (6 GB) and [6].

Baseline. We choose the state-of-the-art backbone solver MiniBones as our baseline (ported to support newer C++ compilers available at `https://github.com/arminbiere/minibones`). As suggested by [18], we use the configuration "`-e -c 100 -i`" (called minibones-core-based), which adopts the core-based approach with a fixed chunk size of 100 and inserts found backbone literals into the input formula explicitly. Additionally, to evaluate how our algorithm improves upon the Algorithm 5 in [18], we choose "`-u -c 100 -i`" (called minibones-iterative) which implements the algorithm and uses activation literals.

Platform. For benchmark collection we used a machine with an AMD Ryzen Threadripper 3970X 32-Core Processor at 4.5 GHz and 256 GB RAM. All other experiments were conducted in parallel on a cluster consisting of 32 machines, each with two 8-core Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz (turbo-mode disabled) and 128 GB RAM. Each instance is allocated to one core with a timeout of 5,000 seconds and a memory limit of 7 GB.

Data Availability. Experimental data including source code and log files are available on `https://cca.informatik.uni-freiburg.de/cadiback`.

Overall Results. We run both CadiBack and the baseline MiniBones on all benchmarks in our benchmark set. We consider an instance solved if the tool completes backbone computation, i.e., classifies all literals as either backbone or non-backbone. The number of instances solved over time are presented in Figure 2. It turns out that the best performing default configuration (default) of CadiBack can solve 732 instances in total, which is 274 more (59.82%) than the best performing configuration of MiniBones, i.e., the iterative configuration minibones-iterative, which solves only 458 instances. Note that, 11 failing runs of CadiBack and 61 failing runs of MiniBones hit the memory limit. It is also instructive to observe that over all selected 1798 instances CadiBack was able to find the first model in 1573 cases, while MiniBones did so in only 1152 cases, which clearly shows the advantage of using CaDiCaL [4] versus MiniSat [13] in this application. It might be interesting to investigate whether this improvement transfers to other applications using MiniSat.

In addition, following the SAT manifesto v1.0 [8], we also compare the default configuration of CadiBack with the best configuration of MiniBones on all satisfiable benchmarks in the main track of SAT competitions from 2020 to 2022. As a result, CadiBack/MiniBones solved 22/9 instances in 2020, 52/17 in 2021 and 41/10 in 2022.

Configurations. We study the impact of different design options in CadiBack by evaluating 12 configurations (see Figure 2) including an extension implementing the core-based approach of MiniBones (Algorithm 7 in [18]). Firstly, we observe that the effects of using smaller chunks were detrimental in our experiments. In fact, the infinite chunk size $K = \infty$ (default) has been very beneficial which solves 732 instances, while chunk size $K = 10$ (chunking) solves only 702 instances and chunk size $K = 1$ (one-by-one) even only 692.

Secondly, we study the impact of design options related to flipping. The experimental results indicate that removing flippable literals from the candidate list (no-flip) does not have a significant overall impact. This result differs from the one given by the authors of MiniBones where the model rotation was detrimental. We attribute this to the efficiency of using watch lists for the flippable check. The really-flip configuration uses "`flip`" and simply tries to flip literals of the candidate chunk in an arbitrary order. It performs similar to the default configuration which uses "`flippable`", but is better in the aspect that the default only found 30,780,841 flippable literals in total, while really-flip found 32,488,468. This directly leads to a reduction of the total number of SAT solver calls, which goes down from 2,070,166 calls in no-flip to 1,478,160 calls in default and even down to 992,404 calls in really-flip.

Thirdly, to evaluate the impact of CaDiCal on CadiBack in detail, including its more advanced inprocessing and its most recent "`constrain`" API to support single clause assumptions [16]. We observe that disabling the inprocessing in CaDiCal (no-inprocessing) significantly degrades the performance from solving 732 instances to 690 instances. Disabling the single clause assumption support from CaDiCal in configuration no-constrain and falling back to activation literals (as MiniBones does) degrades the efficiency of CadiBack even more significantly to solving only 672 instances.

Lastly, we evaluate the impact of our core-based algorithm. The core-based preprocessing in CadiBack only solves 672 instances. However, since the core-based approach falls back to default if the considered literal set becomes empty after removing failed assumptions (see Algorithm 7 in [18] for details), our cores version is more sophisticated than MiniBones (minibones-core-based), thanks to its advanced features in default. In contrast, the core-based MiniBones configuration (minibones-core-based) is slightly better than its iterative version (minibones-iterative) for shorter run times, which matches observations of [18] with the lower time limit of 800 seconds. One can argue, that the reason probably is that the core-based algorithm in MiniBones can rely on literal assumptions [13] avoiding the overhead inflicted from adding temporary clauses and activation literals. However, this slight advantage degrades for long running instances, as can be seen in Figure 2.

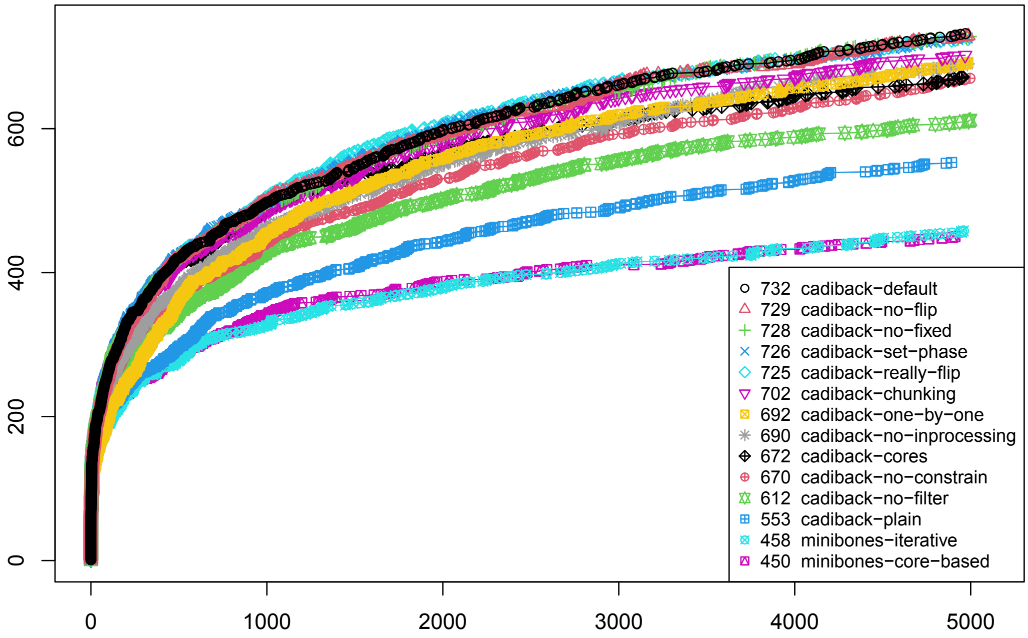


Figure 2 Benchmarks solved (vertical) over time in seconds (horizontal) where backbone extraction completed within 5,000 seconds by 12 CadiBack configurations: default denoting all optimizations enabled except for chunking and cores; no-flip denoting no model rotation; no-fixed representing no checking on candidates for being fixed explicitly; set-phase denoting picking decisions in SAT solver to falsify backbone candidates; really-flip denoting flipping flippable literals eagerly; chunking representing the fine-grained chunk size control ($K = 10$); one-by-one denoting single literal chunks ($K = 1$); no-inprocessing representing no SAT solver internal inprocessing; cores denoting core-based preprocessing; no-constrain meaning only using activation literals instead of using "`constrain`" API; no-filter disables filtering backbone candidates by the disagreement condition; and plain setting $K = 1$ (as one-by-one) and disabling all other optimizations. We also considered 2 MiniBones configurations: iterative implementing Algorithm 5 in [18]; and core-based implementing Algorithm 7 in [18].

7 Conclusion

We revisited backbone algorithms and implemented a new open-source backbone extraction tool CadiBack based on an extended version of the state-of-the-art SAT solver CaDiCaL. Our extensive evaluation on a large set of benchmarks shows a substantial performance improvement by solving 60% more benchmarks than the state-of-the-art MiniBones.

References

1 Anton Belov and João Marques-Silva. Accelerating MUS extraction with recursive model rotation. In Per Bjesse and Anna Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 – November 02, 2011*, pages 37–40. FMCAD Inc., 2011.

2 Armin Biere. Adaptive restart strategies for conflict driven SAT solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing – SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer, 2008. doi:10.1007/978-3-540-79719-7_4.

3 Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, pages 10–11. University of Helsinki, 2022.

4 Armin Biere, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba entering the SAT Competition 2021. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2021 – Solver and Benchmark Descriptions*, volume B-2021-1 of *Department of Computer Science Report Series B*, pages 10–13. University of Helsinki, 2021.

5 Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, 2015. doi:10.1007/978-3-319-24318-4_29.

6 Armin Biere, Nils Froleyks, and Wenxi Wang. Sampled and Normalized Satisfiable Instances from the main track of the SAT Competition 2004 to 2022, March 2023. doi:10.5281/zenodo.7750076.

7 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 391–435. IOS Press, 2021. doi:10.3233/FAIA200992.

8 Armin Biere, Matti Järvisalo, Daniel Le Berre, Kuldeep S. Meel, and Stefan Mengel. The SAT practitioner's manifesto, September 2020. doi:10.5281/zenodo.4500928.

9 Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing – SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2010. doi:10.1007/978-3-642-14186-7_6.

10 Sharlee Climer and Weixiong Zhang. Searching for backbones and fat: A limit-crossing approach with applications. In *AAAI/IAAI*, pages 707–712, 2002.

11 Michael Codish, Yoav Fekete, and Amit Metodi. Backbones for equality. In Valeria Bertacco and Axel Legay, editors, *Hardware and Software: Verification and Testing – 9th International Haifa Verification Conference, HVC 2013, Haifa, Israel, November 5-7, 2013, Proceedings*, volume 8244 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2013. doi:10.1007/978-3-319-03077-7_1.

12 Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *IJCAI*, volume 1, pages 248–253, 2001.

13 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi:10.1007/978-3-540-24605-3_37.

14 Mohamed El Bachir Menaï. A two-phase backbone-based search heuristic for partial MAX-SAT–an initial investigation. In *Innovations in Applied Artificial Intelligence: 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2005, Bari, Italy, June 22-24, 2005. Proceedings 18*, pages 681–684. Springer, 2005.

15 Katalin Fazekas, Armin Biere, and Christoph Scholl. Incremental inprocessing in SAT solving. In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 136–154. Springer, 2019. doi:10.1007/978-3-030-24258-9_9.

16 Nils Froleyks and Armin Biere. Single clause assumption without activation literals to speed-up IC3. In *Formal Methods in Computer Aided Design, FMCAD 2021*, pages 72–76. IEEE, 2021. doi:10.34727/2021/isbn.978-3-85448-046-4_15.

17 Mikoláš Janota. *SAT solving in interactive configuration.* PhD thesis, University College Dublin, 2010.

18 Mikoláš Janota, Inês Lynce, and João Marques-Silva. Algorithms for computing backbones of propositional formulae. *AI Commun.*, 28(2):161–177, 2015. doi:10.3233/AIC-140640.

19 Andreas Kaiser and Wolfgang Küchlin. Detecting inadmissible and necessary variables in large propositional formulae. In *Intl. Joint Conf. on Automated Reasoning (Short Papers)*, pages 96–102. University of Siena, 2001.

20 Philip Kilby, John Slaney, Sylvie Thiébaux, Toby Walsh, et al. Backbones and backdoors in satisfiability. In *AAAI*, volume 5, pages 1368–1373, 2005.

21 D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo. Software fault interactions and implications for software testing. *IEEE Trans. Software Eng.*, 30(6):418–421, 2004. doi:10.1109/TSE.2004.24.

22 João Marques-Silva, Mikoláš Janota, and Inês Lynce. On computing backbones of propositional theories. In Helder Coelho, Rudi Studer, and Michael J. Wooldridge, editors, *ECAI 2010 – 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 15–20. IOS Press, 2010.

23 Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400(6740):133–137, 1999.

24 Andrew J. Parkes. Clustering at the phase transition. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA*, pages 340–345. AAAI Press / The MIT Press, 1997.

25 Alessandro Previti and Matti Järvisalo. A preference-based approach to backbone computation with application to argumentation. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 896–902, 2018.

26 Miroslav N Velev. Formal verification of vliw microprocessors with speculative execution. In *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12*, pages 296–311. Springer, 2000.

27 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1173–1178. Morgan Kaufmann, 2003.

28 Guoqiang Zeng, Chongwei Zheng, Zhengjiang Zhang, and Yongzai Lu. An backbone guided extremal optimization method for solving the hard maximum satisfiability problem. In *2012 International Conference on Computer Application and System Modeling*, pages 1301–1304. Atlantis Press, 2012.

29 Weixiong Zhang. Configuration landscape analysis and backbone guided local search.: Part i: Satisfiability and maximum satisfiability. *Artificial Intelligence*, 158(1):1–26, 2004.

30 Weixiong Zhang. Phase transitions and backbones of the asymmetric traveling salesman problem. *Journal of Artificial Intelligence Research*, 21:471–497, 2004.

31 Weixiong Zhang, Ananda Rangan, Moshe Looks, et al. Backbone guided local search for maximum satisfiability. In *IJCAI*, volume 3, pages 1179–1186, 2003.

32 Yueling Zhang, Min Zhang, and Geguang Pu. Optimizing backbone filtering. *Science of Computer Programming*, 187:102374, 2020.

33 Yueling Zhang, Min Zhang, Geguang Pu, Fu Song, and Jianwen Li. Towards backbone computing: A greedy-whitening based approach. *AI Communications*, 31(3):267–280, 2018.

34 Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 63–66. IEEE, 2011.

35 Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. Silicon fault diagnosis using sequence interpolation with backbones. In Yao-Wen Chang, editor, *The IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2014, San Jose, CA, USA, November 3-6, 2014*, pages 348–355. IEEE, 2014. doi:10.1109/ICCAD.2014.7001373.

36 Charlie Shucheng Zhu, Georg Weissenbacher, Divjyot Sethi, and Sharad Malik. SAT-based techniques for determining backbones for post-silicon fault localisation. In *2011 IEEE International High Level Design Validation and Test Workshop*, pages 84–91. IEEE, 2011.

A Appendix

This appendix provides more experimental details. The left plot in Figure 3 emphasizes why the most simplistic backbone algorithm, i.e., assuming the negation of exactly one remaining backbone candidate literal, does not scale, as it just takes way too many SAT calls.

Furthermore, in a number of applications it can be beneficial to get the backbones as soon as they are found, particularly if the backbone search does not terminate. To that end we evaluate MiniBones and CadiBack as anytime algorithms and compare the number of backbones they find over time. We modified the default configuration of MiniBones (minibones-core-based, i.e., corresponding to options "`-e -i -c 100`") to print a backbone as soon as it is found and evaluated it against the default version of CadiBack on the 2022 SAT competition benchmark set. The results are presented on the right in Figure 3.

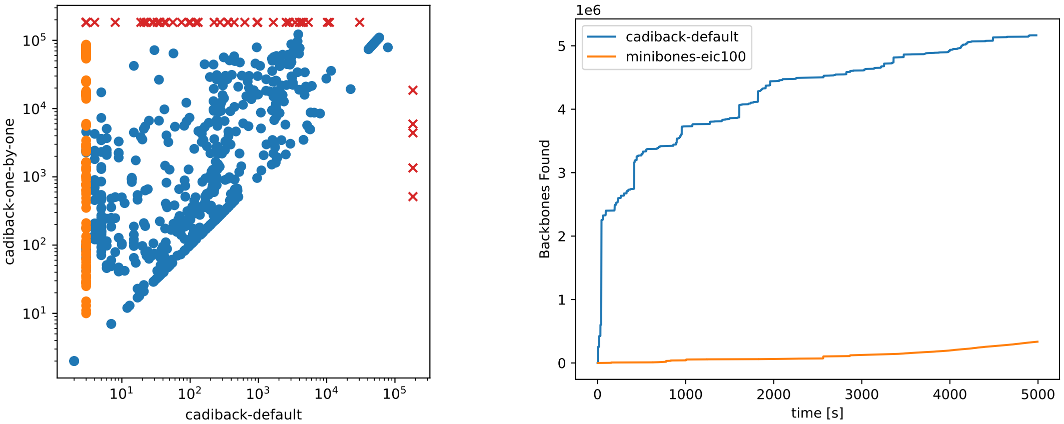


Figure 3 The *left plot* compares the one-by-one and the default configuration. Timeouts for one of the configurations are marked in the margin. Highlighted on the left are 133 (out of 1798) benchmarks that have exactly one model (every variable is in the backbone). Using an infinite chunk size (the default), such benchmarks are always solved in 3 SAT calls. The *right plot* compares MiniBones and CadiBack in an anytime setting. Shown are the number of backbones found combined across all instances in the SAT competition 2022 benchmark set.

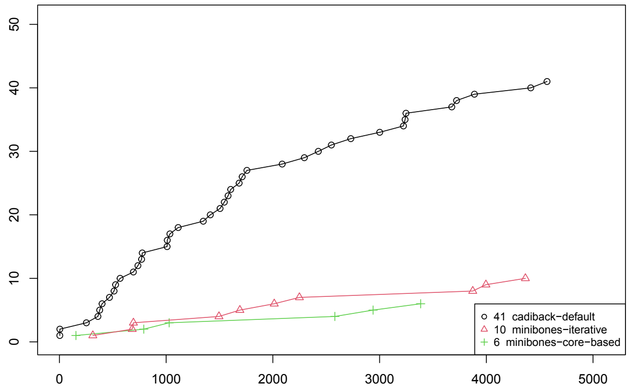


Figure 4 Benchmarks solved on satisfiable instances from the SAT Competition 2022.

Table 1 More detailed results for the runs plotted in Fig. 2 on the large SAT competition 2004–2022 benchmark set where: solved instances; failed to solved; to time out of 5,000 seconds hits; mo memory limit of 7 GB hit; time accumulated process time of solved instances (in seconds); space sum of the maximum memory usage over solved instances (in MB); max maximum memory usage on solved instances (in MB); best number of instances with best shortest solving time; unique uniquely solved number of instances. For the description of the configurations see caption of Fig. 2.

	solved	failed	to	mo	time	space	max	best	unique
cadiback-default	732	842	831	11	694 027	110 614	2600	53	1
cadiback-no-flip	729	845	837	8	686 832	103 021	2600	58	0
cadiback-no-fixed	728	846	835	11	682 242	106 129	2600	70	2
cadiback-set-phase	726	848	838	10	657 492	108 737	2565	163	4
cadiback-really-flip	725	849	838	11	640 447	105 963	2600	46	1
cadiback-chunking	702	872	861	11	630 633	93 625	2600	108	0
cadiback-one-by-one	692	882	871	11	715 101	86 152	2600	30	0
cadiback-no-inprocessing	690	884	873	11	688 418	93 947	2628	116	7
cadiback-cores	672	902	891	11	570 724	100 362	2600	78	1
cadiback-no-constrain	670	904	890	14	693 284	93 836	2546	41	0
cadiback-no-filter	612	962	951	11	562 853	72 360	2600	9	0
cadiback-plain	553	1021	1010	11	544 688	58 500	2655	13	0
minibones-iterative	458	1340	1279	61	402 645	110 138	5281	54	17
minibones-core-based	450	1348	1283	65	348 856	72 793	3542	52	2

Finally we show in Figure 4 the performance of the default version of CadiBack versus the iterative and core-based versions of MiniBones on the last SAT Competition 2022. Table 1 gives more details about the runs plotted in Fig. 2.

QCDCL vs QBF Resolution: Further Insights

Benjamin Böhm
Friedrich-Schiller-Universität Jena, Germany

Olaf Beyersdorff
Friedrich-Schiller-Universität Jena, Germany

Abstract

We continue the investigation on the relations of QCDCL and QBF resolution systems. In particular, we introduce QCDCL versions that tightly characterise QU-Resolution and (a slight variant of) long-distance Q-Resolution. We show that most QCDCL variants – parameterised by different policies for decisions, unit propagations and reductions – lead to incomparable systems for almost all choices of these policies.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases QBF, CDCL, resolution, proof complexity, simulations, lower bounds

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.4

Related Version *Extended Version*: https://eccc.weizmann.ac.il/report/2023/051/

Funding *Olaf Beyersdorff*: Carl-Zeiss Foundation and DFG grant BE 4209/3-1.

1 Introduction

SAT solving has revolutionised the way we practically handle computationally complex problems [29] and emerged as a central tool for numerous applications [15]. Modern SAT solving crucially relies on the paradigm of conflict-driven clause learning (CDCL) [24], on which almost all current SAT solvers are based.

The main theoretical approach to understanding the success of SAT solving (and its limits) comes through proof complexity [19]. From seminal results [1, 5, 26] we know that CDCL – viewed as a non-deterministic procedure – is exactly as powerful as propositional resolution, which is by far the best-understood propositional proof system [19, 23]. However, we also know that practical CDCL using e.g. VSIDS is exponentially weaker than resolution [30]. Moreover, any deterministic CDCL algorithm will be strictly weaker than resolution unless P=NP [2]. In any case, the mentioned results of [1, 5, 26] imply that all formulas hard for resolution will be intractable for modern CDCL solvers (at least when disabling preprocessing).

Solving of quantified Boolean formulas (QBF) extends the success of SAT solving to the presumably computationally harder case of deciding QBFs, a PSPACE-complete problem. While QBF solving utilises quite different algorithmic approaches [14], which build on different proof systems, one of the central paradigms again rests on CDCL, lifted to QBFs in form of QCDCL [31]. In comparison to the propositional case, the main changes are (i) different decision strategies using information from the prefix, (ii) differently implemented unit propagation incorporating universal reductions (i.e., dropping trailing universal variables in clauses), and (iii) adapted methods for learning clauses using a QBF resolution system called long-distance Q-Resolution [3].

The advances in QBF solving have also stimulated growing research in QBF proof complexity [6, 9, 11]. As in the propositional case, QBF resolution systems have received great attention. However, in QBF there are a number of conceptually different resolution systems of varying strength [4, 8, 12]. The core system is Q-Resolution, introduced in 1995 in [22]. This system generalises propositional resolution to QBF by using the resolution rule

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 4; pp. 4:1–4:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for existential pivots and handling universal variables by universal reduction. A stronger calculus QU-Resolution [28] also allows universal pivots in resolution steps (and this is perhaps the most natural QBF resolution system from a logical perspective [7, 9]). Yet another generalisation is provided in the form of long-distance Q-Resolution [3] which allows certain merging steps forbidden in Q-Resolution. As mentioned above, QCDCL traces can be efficiently transformed into long-distance Q-Resolution proofs and this was in fact the reason for creating that proof system.

A recent line of research has aimed at understanding the precise relationship between QCDCL and QBF resolution [10, 16–18, 20]. The findings so far reveal both similarities to the tight relation between CDCL and resolution in SAT as well as crucial differences. While the first work [20] by Janota on this topic showed that practical (deterministic) QCDCL is exponentially weaker than Q-Resolution, the paper [10] demonstrated that QCDCL – even in its non-deterministic version – is incomparable to Q-resolution. This also implies that (non-deterministic) QCDCL is exponentially weaker than long-distance Q-Resolution. This is in sharp contrast to the equivalence of SAT and resolution in the propositional case [1, 5, 26], as explained above.

These results were strengthened in [16] by developing a lower-bound technique for QCDCL via a new notion of gauge, by which a number of lower bounds for QCDCL can be demonstrated (which not necessarily hinge on any QBF resolution hardness). Further, [17, 18] showed that several QCDCL variants, utilising e.g. cube learning, pure-literal elimination, and different decision strategies give rise to proof systems of different strength.

1.1 Our contributions

In this paper we continue this recent line of research to try to understand to precisely determine the relationship of QCDCL variants and different QBF resolution systems. The central quest of our research here is to find different QCDCL variants that are as strong as QU-Resolution and long-distance Q-Resolution. While we do not claim that these new algorithms are of immediate practical interest, we believe it is important to theoretically gauge the full potential of QCDCL. Our results can be summarised as follows.

(a) New QCDCL versions. We realise that there are at least three crucial QCDCL components that determine the strength of the algorithm. These are (i) whether decisions are made according to the prefix or not (policies LEV-ORD or ANY-ORD), (ii) whether unit propagation always or never includes universal reduction (policies ALL-RED, NO-RED) or whether this can be freely chosen at each propagation (ANY-RED), and (iii) whether unit propagation can propagate only existential variables (as in practical QCDCL, policy EXI-PROP) or whether also universal variables can be propagated (ALL-PROP).

While some of these policies were already defined and investigated in earlier works [10, 17, 18], the policies ANY-RED and ALL-PROP are considered here for the first time. We note that a solver implementing the strategy ALL-PROP together with LEV-ORD and NO-RED was recently presented by Slivovsky [27] (in fact this motivated our definition of the policies EXI-PROP and ALL-PROP). We demonstrate that in principle, all the aforementioned policies can be combined to yield sound and complete QCDCL algorithms (Proposition 8). We denote these as e.g. $\mathsf{QCDCL}^{\textsc{Lev-Ord}}_{\textsc{All-Red},\textsc{Exi-Prop}}$ (this combination models standard QCDCL).

(b) Characterisation of QBF proof systems. In our main result we tightly characterise the proof systems QU-Resolution by $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{No-Red},\textsc{All-Prop}}$ as well as (a slight variant of) long-distance Q-Resolution by $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{Any-Red},\textsc{Exi-Prop}}$ (Proposition 18 and Theorem 25). These results are similar in spirit (and proof method) to the characterisation of propositional resolution by CDCL [26] and Q-Resolution by $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{No-Red},\textsc{Exi-Prop}}$ [10]. However, quite some technical care is needed for the simulations to go through with the modified policies, for which we use the new notion of a blockade (Definition 22).

The mentioned variant of long-distance Q-Resolution – called mLD-Q-Res (for modified long-distance Q-Resolution, Definition 17) – is defined such as to contain exactly those steps that are needed for clause learning in standard QCDCL. The original definition of long-distance Q-resolution also allows some merging steps that do not occur in clause learning (those that have merged literals left of the pivot in both clauses). We leave open whether mLD-Q-Res is indeed weaker or equivalent to long-distance Q-Resolution (cf. Section 6).

(c) Separations between QCDCL variants. We clarify the joint simulation order of QBF resolution and QCDCL systems (cf. Figure 1 for an overview depicting known and new results). In general, the emerging picture shows that different choices of policies lead to incomparable systems (and could thus in principle be exploited for gains in practical solving over currently used QCDCL, cf. [18, 27]).

One set of results that we highlight here concerns the new system $\mathsf{QCDCL}^{\textsc{Lev-Ord}}_{\textsc{Any-Red},\textsc{Exi-Prop}}$, which we show to be strictly stronger than standard QCDCL, yet still weaker than mLD-Q-Res (and incomparable to Q-Resolution). To show that the system is strictly stronger than standard QCDCL (= $\mathsf{QCDCL}^{\textsc{Lev-Ord}}_{\textsc{All-Red},\textsc{Exi-Prop}}$), we exhibit some new family of QBFs which we show to be hard under the All-Red or No-Red policies, yet tractable under Any-Red.

1.2 Organisation

The remainder of this paper is organised as follows. We start by reviewing some notions from QBFs and QBF resolution systems in Section 2. In Section 3 we review the existing QCDCL models and define our variants. In Section 4 we investigate the simulation order of the QCDCL proof systems and show various separations. In Section 5 we obtain our main results, the characterisation of the proof systems QU-Res and mLD-Q-Res. We conclude in Section 6 with some open questions.

2 Preliminaries

Propositional and quantified formulas. Variables x and negated variables $\bar{x}$ are called *literals*. We denote the corresponding variable as $\text{var}(x) := \text{var}(\bar{x}) := x$.

A *clause* is a disjunction of literals, but we will sometimes interpret them as sets of literals on which we can perform set-theoretic operations. A *unit clause* (ℓ) is a clause that consists of only one literal. The *empty clause* consists of zero literals, denoted $(\bot)$. We sometimes interpret $(\bot)$ as a unit clause with the "empty literal" $\bot$. A clause C is called *tautological* if $\{\ell, \bar{\ell}\} \subseteq C$ for some literal ℓ. Alternatively, we will sometimes write $\ell^* \in C$ instead of $\{\ell, \bar{\ell}\} \subseteq C$.

A *cube* is a conjunction of literals and can also be viewed as a set of literals. We define a *unit cube* of a literal ℓ, denoted by $[\ell]$, and the *empty cube* $[\top]$ with "empty literal" $\top$. A cube D is *contradictory* if $\{\ell, \bar{\ell}\} \subseteq D$ for some literal ℓ. If C is a clause or a cube, we define $\text{var}(C) := \{\text{var}(\ell) : \ell \in C\}$. The negation of a clause $C = \ell_1 \vee \ldots \vee \ell_m$ is the cube $\neg C := \overline{C} := \bar{\ell}_1 \wedge \ldots \wedge \bar{\ell}_m$.

A *(total) assignment* σ of a set of variables V is a non-tautological set of literals such that for all $x \in V$ there is some $\ell \in \sigma$ with $\text{var}(\ell) = x$. A *partial assignment* σ of V is an assignment of a subset $W \subseteq V$. A clause C is *satisfied* by an assignment σ if $C \cap \sigma \neq \emptyset$.

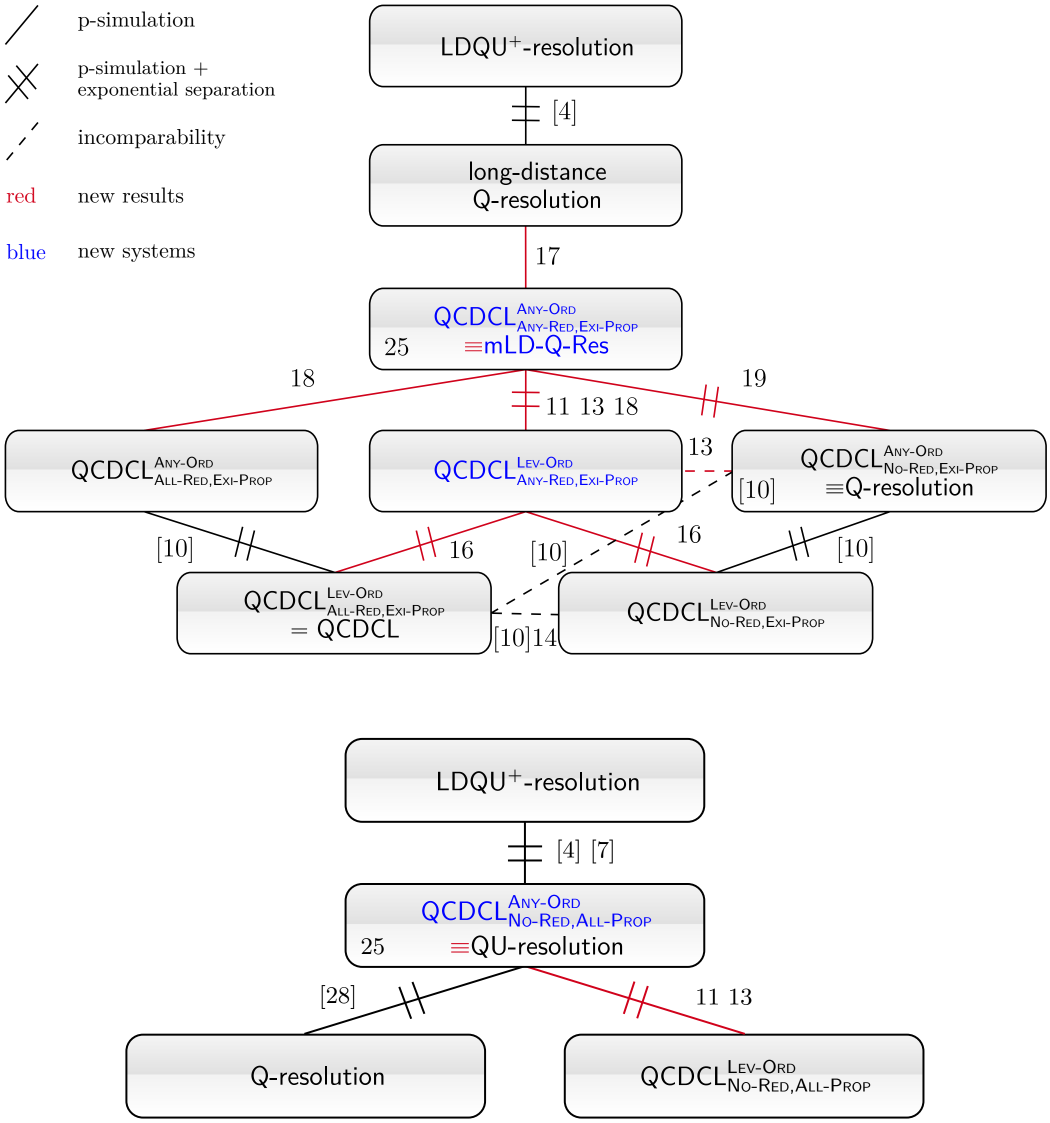

p-simulation
p-simulation + exponential separation
incomparability
red new results
blue new systems
LDQU$^+$-resolution
[4]
long-distance Q-resolution
17
$\text{QCDCL}^{\text{Any-Ord}}_{\text{Any-Red,Exi-Prop}}$
25 ≡mLD-Q-Res
18
11 13 18
19
$\text{QCDCL}^{\text{Any-Ord}}_{\text{All-Red,Exi-Prop}}$
$\text{QCDCL}^{\text{Lev-Ord}}_{\text{Any-Red,Exi-Prop}}$
13
$\text{QCDCL}^{\text{Any-Ord}}_{\text{No-Red,Exi-Prop}}$
[10] ≡Q-resolution
[10]
16
[10]
16
[10]
$\text{QCDCL}^{\text{Lev-Ord}}_{\text{All-Red,Exi-Prop}}$
= QCDCL
[10]14
$\text{QCDCL}^{\text{Lev-Ord}}_{\text{No-Red,Exi-Prop}}$
LDQU$^+$-resolution
[4] [7]
$\text{QCDCL}^{\text{Any-Ord}}_{\text{No-Red,All-Prop}}$
25 ≡QU-resolution
[28]
11 13
Q-resolution
$\text{QCDCL}^{\text{Lev-Ord}}_{\text{No-Red,All-Prop}}$

Figure 1 Hasse diagrams of the simulation order of QCDCL with EXI-PROP (above) and ALL-PROP (below) plus corresponding proof systems. Blue names represent new systems introduced here. Numbers in brackets are external references, while numbers without brackets are lemmas, propositions or theorems of this paper.

A cube D is *falsified* by σ if $\neg D \cap \sigma \neq \varnothing$. A clause C that is not satisfied by σ can be *restricted* by σ, defined as $C|_\sigma := \bigvee_{\ell \in C, \bar{\ell} \notin \sigma} \ell$. Similarly we can restrict a non-falsified cube D as $D|_\sigma := \bigwedge_{\ell \in D \setminus \sigma} \ell$. Intuitively, an assignment sets all its literals to true.

A *CNF* (conjunctive normal form) is a conjunction of clauses and a *DNF* (disjunctive normal form) is a disjunction of cubes. We restrict a CNF (resp. DNF) ϕ by an assignment σ as $\phi|_\sigma := \bigwedge_{C \in \phi \text{ non-satisfied}} C|_\sigma$ (resp. $\phi|_\sigma := \bigvee_{D \in \phi \text{ non-falsified}} D|_\sigma$). For a CNF (DNF) ϕ and an assignment σ, if $\phi|_\sigma = \varnothing$, then ϕ is *satisfied* (*falsified*) by σ.

A *QBF* (quantified Boolean formula) $\Phi = \mathcal{Q} \cdot \phi$ consists of a propositional formula ϕ, called the *matrix*, and a *prefix* $\mathcal{Q}$. A *prefix* $\mathcal{Q} = \mathcal{Q}'_1 V_1 \ldots \mathcal{Q}'_s V_s$ consists of non-empty and pairwise disjoint sets of variables $V_1, \ldots, V_s$ and quantifiers $\mathcal{Q}'_1, \ldots, \mathcal{Q}'_s \in \{\exists, \forall\}$ with $\mathcal{Q}'_i \neq \mathcal{Q}'_{i+1}$ for $i \in [s-1]$. For a variable x in $\mathcal{Q}$, the *quantifier level* is $\text{lv}(x) := \text{lv}_\Phi(x) := i$, if $x \in V_i$. For $\text{lv}_\Phi(\ell_1) < \text{lv}_\Phi(\ell_2)$ we write $\ell_1 <_\Phi \ell_2$, while $\ell_1 \leqslant_\Phi \ell_2$ means $\ell_1 <_\Phi \ell_2$ or $\ell_1 = \ell_2$.

For a QBF $\Phi = \mathcal{Q} \cdot \phi$ with ϕ a CNF, we call Φ a *QCNF*. We define $\mathfrak{C}(\Phi) := \phi$. The QBF Φ is an *AQBF* (augmented QBF), if $\phi = \psi \vee \chi$ with CNF ψ and DNF χ. Again we write $\mathfrak{C}(\Phi) := \psi$ and $\mathfrak{D}(\Phi) := \chi$. We will sometimes interpret QCNFs as sets of clauses and cubes. If Φ is a QCNF or AQBF, we define $\text{var}(\Phi) := \bigcup_{C \in \Phi} \text{var}(C)$.

We restrict a QCNF $\Phi = \mathcal{Q} \cdot \phi$ by an assignment σ as $\Phi|_\sigma := \mathcal{Q}|_\sigma \cdot \phi|_\sigma$, where $\mathcal{Q}|_\sigma$ is obtained by deleting all variables from $\mathcal{Q}$ that appear in σ. Analogously, we restrict an AQBF $\Phi = \mathcal{Q} \cdot (\psi \vee \chi)$ as $\Phi|_\sigma := \mathcal{Q}|_\sigma \cdot (\psi|_\sigma \vee \chi|_\sigma)$.

If L is a set of literals (e.g., an assignment), we can get the negation of L, which we define as $\neg L := \overline{L} := \{\bar{\ell} |\, \ell \in L\}$.

(Long-distance) Q-resolution. Let C_1 and C_2 be two clauses from a QCNF or AQBF Φ. Let ℓ be an existential literal with $\text{var}(\ell) \notin \text{var}(C_1) \cup \text{var}(C_2)$. The *resolvent* of $C_1 \vee \ell$ and $C_2 \vee \bar{\ell}$ over ℓ is defined as

$$(C_1 \vee \ell) \overset{\ell}{\bowtie}_\Phi (C_2 \vee \bar{\ell}) := C_1 \vee C_2$$

Let $C := \ell_1 \vee \ldots \vee \ell_m$ be a clause from a QCNF or AQBF Φ such that $\ell_i \leqslant_\Phi \ell_j$ for all $i < j$, while $i, j \in [m]$. Let k be minimal such that $\ell_k, \ldots, \ell_m$ are universal. Then we can perform a *universal reduction* step and obtain

$$\text{red}^\forall_\Phi(C) := \ell_1 \vee \ldots \vee \ell_{k-1}.$$

If it is clear that C is a clause, we can just write $\text{red}_\Phi(C)$ or even $\text{red}(C)$, if the QBF Φ is also obvious. We will write $\text{red}(\Phi) = \text{red}_\Phi(\Phi)$, if we reduce all clauses of Φ according to its prefix.

We can also perform *partial universal reduction*. Let K is a non-tautological set of literals and let $C := \ell_1 \vee \ldots \vee \ell_m$ be a clause from a QCNF Φ such that

$$\{\ell_k, \ldots, \ell_m\} = \{\ell \in C |\, \ell \in K,\ \ell \text{ is universal and } x <_\Phi \ell \text{ for all existential } x \in C\}.$$

Then we can partially reduce C by K and obtain

$$\text{red}^\forall_{\Phi,K}(C) := \ell_1 \vee \ldots \vee \ell_{k-1}.$$

Intuitively, we will reduce all reducible literals that are also contained in K.

As before, we simply write red_K instead of $\text{red}^\forall_{\Phi,K}$ if the context is clear.

As defined by Kleine Büning et al. [22], a Q-resolution proof π from a QCNF or AQBF Φ of a clause C is a sequence of clauses $\pi = (C_i)_{i=1}^m$, such that $C_m = C$ and for each C_i one of the following holds:

- *Axiom:* $C_i \in \mathfrak{C}(\Phi)$;
- *Resolution:* $C_i = C_j \overset{x}{\bowtie}_\Phi C_k$ with x existential, $j, k < i$, and C_i non-tautological;
- *Reduction:* $C_i = \text{red}^\forall_\Phi(C_j)$ for some $j < i$.

[3] introduced an extension of Q-resolution proofs to long-distance Q-resolution proofs by replacing the resolution rule by

- *Resolution (long-distance):* $C_i = C_j \overset{x}{\bowtie} C_k$ with x existential and $j, k < i$. The resolvent C_i is allowed to contain tautologies such as $u \vee \bar{u}$, if u is universal. If there is such a universal $u \in \text{var}(C_j) \cap \text{var}(C_k)$, then we require $x <_\Phi u$.

The work [28] presented a further extension for Q-resolution, called QU-resolution, where we can also resolve over universal literals. Formally, it replaces the resolution rule by

- *Resolution (QU-Res)*: $C_i = C_j \overset{x}{\bowtie}_\Phi C_k$ with x existential *or* universal, $j, k < i$, and C_i non-tautological.

In [4], long-distance Q-resolution and QU-resolution were combined into a new proof system: long-distance QU$^+$-resolution. The resolution rule is as follows:

- *Resolution (long-distance QU$^+$-Res):* $C_i = C_j \overset{x}{\bowtie} C_k$ with x existential *or* universal and $j, k < i$. The resolvent C_i is allowed to contain tautologies such as $u \vee \bar{u}$, if u is universal. If there is a such a universal $u \in \text{var}(C_j) \cap \text{var}(C_k)$, then we require $\text{index}(x) < \text{index}(u)$, where index( ) is a fixed total order on the variables of Φ such that $\text{index}(a_1) < \text{index}(a_2)$ whenever $a_1 <_\Phi a_2$ for variables a_1, a_2 of Φ.

A Q-resolution (resp. long-distance Q-resolution, QU-resolution or long-distance QU$^+$-resolution) proof from Φ of the empty clause ($\perp$) is called a *refutation* of Φ. In that case, Φ is called *false*. We will sometimes interpret π as a set of clauses.

For the sake of completeness, we note that the above described proof systems are refutational proof systems that cannot be used to prove the truth of a QBF. For that, we would need analogously defined proof systems that work on cubes instead of clauses. For these proof systems, it is common to use the notion *consensus* instead of resolution, as well as *verification* instead of *refutation*. However, as we will purely concentrate on false formulas in this paper, we omit defining these aspects in more detail.

A proof system P *p-simulates* a system Q, if every Q proof can be transformed in polynomial time into a P proof of the same formula. P and Q are *p-equivalent* (denoted $P \equiv_p Q$) if they p-simulate each other.

3 Our QCDCL models

First, we need to formalise QCDCL procedures as proof systems in order to analyse their complexity. We follow the approach initiated in [10, 16–18].

We store all relevant information of a QCDCL run in *trails*. Since QCDCL uses several runs and potentially also restarts, a QCDCL proof will typically consist of many trails.

▶ **Definition 1** (trails). *A* trail $\mathcal{T}$ *for a QCNF or AQBF Φ is a (finite) sequence of pairwise distinct literals from Φ, including the empty literals $\perp$ and $\top$. Each two literals in $\mathcal{T}$ have to correspond to pairwise distinct variables from Φ. In general, a trail has the form*

$$\mathcal{T} = (p_{(0,1)}, \dots, p_{(0,g_0)}; \mathbf{d_1}, p_{(1,1)}, \dots, p_{(1,g_1)}; \dots; \mathbf{d_r}, p_{(r,1)}, \dots, p_{(r,g_r)}), \tag{1}$$

where the d_i are decision literals *and $p_{(i,j)}$ are* propagated literals*. Decision literals are written in* **boldface***. We use a semicolon before each decision to mark the end of a decision level. If one of the empty literals $\perp$ or $\top$ is contained in $\mathcal{T}$, then it has to be the last literal $p_{(r,g_r)}$. In this case, we say that $\mathcal{T}$ has* run into a conflict*.*

Trails can be interpreted as non-tautological sets of literals, and therefore as (partial) assignments. We write $x <_\mathcal{T} y$ if $x, y \in \mathcal{T}$ and x is left of y in $\mathcal{T}$. Furthermore, we write $x \leqslant_\mathcal{T} y$ if $x <_\mathcal{T} y$ or $x = y$.

As trails are produced gradually from left to right in an algorithm, we define $\mathcal{T}[i, j]$ for $i \in \{0, \dots, r\}$ and $j \in \{0, \dots, g_i\}$ as the subtrail that contains all literals from $\mathcal{T}$ up to (and excluding) $p_{(i,j)}$ (resp. d_i, if $j = 0$) in the same order. Intuitively, $\mathcal{T}[i, j]$ is the state of the trail before we assigned the literal at the point $[i, j]$ (which is $p_{(i,j)}$ or d_i).

For each point $[i, j]$ in the trail there must exist a set of literals $K_{(i,j)}$ which we call the reductive set at point *$[i, j]$. Intuitively, $K_{(i,j)}$ contains all literals that are reduced directly before the point $[i, j]$. The sets $K_{(i,j)}$ depend on the QCDCL variant (i.e., the reduction policy). Note that these sets are non-empty only if reduction is enabled.*

For each propagated literal $p_{(i,j)} \in \mathcal{T}$ there has to be be a clause (or cube) $\mathrm{ante}_{\mathcal{T}}(p_{(i,j)})$ such that $\mathrm{red}_{K_{(i,j)}}(\mathrm{ante}_{\mathcal{T}}(p_{(i,j)})|_{\mathcal{T}[i,j]}) = (p_{(i,j)})$ (or $[p_{(i,j)}]$). We call such a clause (cube) the antecedent clause (cube) *of $p_{(i,j)}$.*

▶ Remark 2. In classic QCDCL, all $K_{(i,j)}$ are set to $\mathrm{var}(\Phi) \cup \overline{\mathrm{var}(\Phi)}$.

We state some general facts about trails and antecedent clauses/cubes.

▶ Remark 3. Let $\mathcal{T}$ be a trail, $\ell \in \mathcal{T}$ a propagated literal and $A := \mathrm{ante}_{\mathcal{T}}(\ell)$.

- If ℓ is existential, then $\ell \in A$ and for each existential literal $x \in A$ with $x \neq \ell$ we need $\bar{x} <_{\mathcal{T}} \ell$.
- If ℓ is universal, then $\bar{\ell} \in A$ and for each universal literal $u \in A$ with $u \neq \bar{\ell}$ we need $u <_{\mathcal{T}} \ell$.

▶ **Definition 4** (natural trails)**.** *We call a trail $\mathcal{T}$* natural *for formula Φ, if for each $i \in \{0, \dots, r\}$ the formula $\mathrm{red}_{K_{(i,0)}}(\Phi|_{\mathcal{T}[i,0]})$, contains unit or empty constraints. Furthermore, the formula $\mathrm{red}_{K_{(i,j)}}\Phi|_{\mathcal{T}[i,j]}$ must not contain empty constraints for each $i \in [r]$, $j \in [g_i]$, except $[i,j] = [r, g_r]$. Intuitively, this means that decisions are only made if there are no more propagations on the same decision level possible. Also, conflicts must be immediately taken care of.*

▶ Remark 5. Although it is allowed to define all sets $K_{(i,j)}$ differently, it might make sense from a practical perspective to weaken these possibilities. We point out three nuances of partial reduction in QCDCL that are interesting to consider:

(i) We change the reductive set after each propagation or decision step. That means that all sets $K_{(i,j)}$ might be different. This is the strongest possible version of partial reduction.

(ii) We only update the reductive set after backtracking. That means the sets $K_{(i,j)}$ are constant for each trail. It will turn out that this version is enough for our characterisation of mLD-Q-Res (cf. Theorem 25). Consequently, this version is as strong as version (i).

(iii) We never change the reductive set. That means that the sets $K_{(i,j)}$ remain constant throughout the whole QCDCL proof. This version is enough for the separation between systems with and systems without partial reduction (cf. Theorem 16).

▶ **Definition 6** (learnable constraints)**.** *Let $\mathcal{T}$ be a trail for Φ of the form (1) with $p_{(r,g_r)} \in \{\bot, \top\}$. Starting with $\mathrm{ante}_{\mathcal{T}}(\bot)$ (resp. $\mathrm{ante}_{\mathcal{T}}(\top)$) we reversely resolve with the antecedent clauses (cubes) that were used to propagate the existential (universal) variables, until we stop at some point. Literals that were propagated via cubes (clauses) will be interpreted as decisions. If a resolution step cannot be performed at some point due to a missing pivot, we simply skip that antecedent. The clause (cube) we so derive is a* learnable constraint*. We denote the sequence of learnable constraints by $\mathfrak{L}(\mathcal{T})$.*

We can also learn cubes from trails that did not run into conflict. If $\mathcal{T}$ is a total assignment of the variables from Φ, then we define the set of learnable constraints as the set of cubes $\mathfrak{L}(\mathcal{T}) := \{\mathrm{red}^{\exists}_{\Phi}(D) |\, D \subseteq \mathcal{T}$ and D satisfies $\mathfrak{C}(\Phi)\}$.

Generally, we allow to learn an arbitrary constraint. However, for the characterisations, it suffices to concentrate on clause learning. Additionally, most of the time we will simply learn the clause which we obtain after propagation over every available literal in the trail. This clause can only consist of negated decision literals, and literals that were reduced during unit propagation. Since this is the last clause we can derive during clause learning in a trail $\mathcal{T}$, we will refer to that clause as the *rightmost clause in $\mathcal{L}(\mathcal{T})$*.

▶ **Definition 7** (QCDCL proof systems). *Let* $\mathsf{D} \in \{$*LEV-ORD*, *ANY-ORD*$\}$ *a decision policy,* $\mathsf{R} \in \{$*ALL-RED*, *NO-RED*, *ANY-RED*$\}$ *a reduction policy and* $\mathsf{P} \in \{$*EXI-PROP*, *ALL-PROP*$\}$ *a propagation policy (all defined below). A* $\mathsf{QCDCL}^{\mathsf{D}}_{\mathsf{R,P}}$ *proof* ι *from a QCNF* $\Phi = \mathcal{Q} \cdot \phi$ *of a clause or cube* C *is a (finite) sequence of triples*

$$\iota := [(\mathcal{T}_i, C_i, \pi_i)]_{i=1}^{m},$$

where $C_m = C$*, each* $\mathcal{T}_i$ *is a trail for* Φ_i *that follows the policies* D*,* R *and* P*, each* $C_i \in \mathfrak{L}(\mathcal{T}_i)$ *is one of the constraints we can learn from each trail and* π_i *is the proof from* Φ_i *of* C_i *we obtain by performing the steps described in Definition 6, where* Φ_i *are AQBFs that are defined recursively by setting* $\Phi_1 := \mathcal{Q} \cdot (\mathfrak{C}(\Phi) \vee \varnothing)$ *and*

$$\Phi_{j+1} := \begin{cases} \mathcal{Q} \cdot ((\mathfrak{C}(\Phi_j) \wedge C_j) \vee \mathfrak{D}(\Phi_j)) & \text{if } C_j \text{ is a clause,} \\ \mathcal{Q} \cdot (\mathfrak{C}(\Phi_j) \vee (\mathfrak{D}(\Phi_j) \vee C_j)) & \text{if } C_j \text{ is a cube,} \end{cases}$$

for $j = 1, \ldots, m-1$*. If necessary, we set* $\pi_i := \varnothing$*.*

We now explain the three types of policies:

- *Decision policies:*
  - *LEV-ORD: For each decision* d_i *we have that* $lv_{\Psi|_{\mathcal{T}[i,0]}}(d_i) = 1$*. I.e., decisions are level-ordered.*
  - *ANY-ORD: Decisions can be made arbitrarily in any order.*
- *Reduction policies:*
  - *ALL-RED: All* $K_{(i,j)}$ *are set to* $var(\Phi) \cup \overline{var(\Phi)}$*. This is the classic setting – we have to reduce all reducible literals during unit propagation.*
  - *NO-RED: All* $K_{(i,j)}$ *are set to* $\varnothing$*. We are not allowed to reduce during unit propagation at all. There is one exception: Combined with* ALL-PROP*, we are allowed (but not forced) to reduce universal unit clauses (existential unit cubes) and immediately obtain a conflict. This is due to reasons of completeness which will be explained later.*
  - *ANY-RED: The sets* $K_{(i,j)}$ *can be set arbitrarily. Hence, we can choose after each propagation or decisions step which literals are to be reduced next.*
- *Propagation policies:*
  - *EXI-PROP: Unit clauses (cubes) can only propagate existential (universal) literals. Universal (existential) unit clauses (cubes) will be reduced to the empty clause (cube) if allowed by the reduction policy.*
  - *ALL-PROP: Universal (existential) unit clauses (cubes) will lead to the propagation of the universal (existential) unit literal. This policy is nullified if combined with* ALL-RED*. If combined with* NO-RED*, we are allowed to reduce universal (existential) unit clauses (cubes) instead of doing a unit propagation. This is due to reasons of completeness.*

Having defined all policies, we can now denote trails that follow the policies D*,* R *and* P *as* $\mathsf{QCDCL}^{\mathsf{D}}_{\mathsf{R,P}}$ trails.

We require that $\mathcal{T}_1$ *is a natural* $\mathsf{QCDCL}^{\mathsf{D}}_{\mathsf{R,P}}$ *trail and for each* $2 \leqslant i \leqslant m$ *there is a point* $[a_i, b_i]$ *such that* $\mathcal{T}_i[a_i, b_i] = \mathcal{T}_{i-1}[a_i, b_i]$ *and* $\mathcal{T}_i \backslash \mathcal{T}_i[a_i, b_i]$ *has to be a natural* $\mathsf{QCDCL}^{\mathsf{D}}_{\mathsf{R,P}}$ *trail for* $\Phi_i|_{\mathcal{T}_i[a_i,b_i]}$*. This process is called* backtracking*. If* $\mathcal{T}_{i-1}[a_i, b_i] = \varnothing$*, then this is also called a* restart*.*

If $C = C_m = (\bot)$*, then* ι *is called a* $\mathsf{QCDCL}^{\mathsf{D}}_{\mathsf{R,P}}$ refutation *of* Φ*. If* $C = C_m = [\top]$*, then* ι *is called a* $\mathsf{QCDCL}^{\mathsf{D}}_{\mathsf{R,P}}$ verification *of* Φ*. The proof ends once we have learned* $(\bot)$ *or* $[\top]$*.*

If C *is a clause, we can stick together the* long-distance Q-resolution *derivations from* $\{\pi_1, \ldots, \pi_m\}$ *and obtain a* long-distance Q-resolution *proof from* Φ *of* C*, which we call* $\mathfrak{R}(\iota)$*.*

The size *of* ι *is defined as* $|\iota| := \sum_{i=1}^{m} |\mathcal{T}_i|$*. Obviously, we have* $|\mathfrak{R}(\iota)| \in \mathcal{O}(|\iota|)$*.*

We can show that all combinations of the above policies lead to sound and complete proof systems (and algorithms).

▶ **Proposition 8.** *All defined QCDCL variants are sound and complete.*

Proof. It suffices to show completeness for the weakest combinations. Hence, we can use LEV-ORD and choose between ALL-RED and NO-RED, as both are subsumed by ANY-RED. For EXI-PROP, completeness was already shown in [10]. For ALL-PROP, we distinguish two cases:

(i) ALL-RED: Then we will never propagate universal (existential) literals via clauses (cubes), as they will always be directly reduced to the empty clause (cube). Hence, this system is the same as if we would have chosen EXI-PROP.

(ii) NO-RED: We are not forced to do universal (existential) propagations via clauses (cubes). Therefore, the version with EXI-PROP is already simulated by this combination system.

The soundness follows from the soundness of long-distance QU^+-resolution (long-distance QU^+-consensus) proofs, which can be extracted from all QCDCL variants defined here. ◀

4 The simulation order of QCDCL proof systems

While the policies ALL-RED and NO-RED were already introduced in work (cf. [10]), in which an incomparability between these two models was shown, it is natural to analyse their relation to our new policy ANY-RED. Obviously, ANY-RED covers (hence: simulates) both ALL-RED an NO-RED, as we can simply choose to reduce everything or nothing. We want to prove now that both ALL-RED and NO-RED are exponentially worse than ANY-RED on some family of QBFs. I.e., we want to show that there exist formulas where we need to reduce *some* but not *all* literals during unit propagation.

These formulas will be hand-crafted, consisting of two already well-known QCNFs, named $\texttt{MirrorCR}_n$, which is a modified version of the Completion Principle [21], and $\texttt{QParity}_n$ [12].

▶ **Definition 9** ([17]). *The QCNF* $\texttt{MirrorCR}_n$ *consists of the prefix* $\exists T \forall u \exists T$, *where* $X := \{x_{(1,1)}, \dots, x_{(n,n)}\}$ *and* $T := \{a_1, \dots, a_n, b_1, \dots, b_n\}$, *and the matrix*

$$\begin{array}{lllll} x_{(i,j)} \vee u \vee a_i & \bar{a}_1 \vee \ldots \vee \bar{a}_n & x_{(i,j)} \vee \bar{u} \vee \bar{a}_i & a_1 \vee \ldots \vee a_n & \\ \bar{x}_{(i,j)} \vee \bar{u} \vee b_j & \bar{b}_1 \vee \ldots \vee \bar{b}_n & \bar{x}_{(i,j)} \vee u \vee \bar{b}_j & b_1 \vee \ldots \vee b_n & \textit{for } i, j \in [n]. \end{array}$$

The reason why we use $\texttt{MirrorCR}_n$ instead of $\texttt{CR}_n$ is because its matrix is unsatisfiable. That means that cube learning, which might have a positive effect on $\texttt{CR}_n$ (note that there are false QCNFs that become easy with cube learning [17]) is now completely unavailable. Additionally, we can now guarantee to always get a conflict once all variables from $\texttt{MirrorCR}_n$ got assigned.

▶ **Lemma 10** ([17]). *The matrix* $\mathfrak{C}(\texttt{MirrorCR}_n)$ *of* $\texttt{MirrorCR}_n$ *is unsatisfiable as a propositional formula.*

As $\texttt{MirrorCR}_n$ is simply an extension of the Completion Principle ($\texttt{CR}_n$), which is known to be easy for Q-resolution [21], we can simply reuse the exact same refutation from [21]. Note that we do not need all axiom clauses to refute the formula.

▶ **Proposition 11** ([17]). *The QBFs* $\texttt{MirrorCR}_n$ *have polynomial-size* Q-resolution *refutations.*

▶ **Definition 12** ([12]). *The QCNF* $\texttt{QParity}_n(Y, w, S)$ *consists of the prefix* $\exists Y \forall w \exists S$, *where* $Y := \{y_1, \dots, y_n\}$ *and* $S := \{s_2, \dots, s_n\}$, *and the matrix*

$$\begin{array}{lllll} y_1 \vee y_2 \vee \bar{s}_2 & y_1 \vee \bar{y}_2 \vee s_2 & \bar{y}_1 \vee y_2 \vee s_2 & \bar{y}_1 \vee \bar{y}_2 \vee \bar{s}_2 & \\ y_i \vee s_{i-1} \vee \bar{s}_i & y_i \vee \bar{s}_{i-1} \vee s_i & \bar{y}_i \vee s_{i-1} \vee s_i & \bar{y}_i \vee \bar{s}_{i-1} \vee \bar{s}_i & \textit{for } i \in \{2, \dots, n\}, \\ s_n \vee w & \bar{s}_n \vee \bar{w}. & & & \end{array}$$

When introduced in [17], it was shown that $\texttt{MirrorCR}_n$ is hard for all QCDCL models with level-ordered decisions considered in [17]. We generalize this result and show that the lower bound for $\texttt{MirrorCR}_n$ indeed only depends on the decision policy used and also holds for our new models introduced here.

▶ **Proposition 13.** *The QBFs* $\texttt{MirrorCR}_n(X, u, T)$ *need exponential-sized refutations in all our QCDCL variants with the* LEV-ORD *policy.*

Proof. (Sketch) We recall the hardness results of $\texttt{MirrorCR}_n$ for classical QCDCL in [17], which were independent of the reduction policy. One can also show that it is impossible to propagate universal literals, therefore the propagation policies do not matter, either. ◀

With the QBFs $\texttt{QParity}_n$ one obtains one direction of the incomparability between classical QCDCL (here called $\mathsf{QCDCL}^{\textsf{Lev-Ord}}_{\textsf{All-Red,Exi-Prop}}$) and Q-resolution, being easy for the former and hard for the latter system.

▶ **Theorem 14** ([10, 13])**.** *The QBFs* $\texttt{QParity}_n$ *need exponential-sized* Q-resolution *and* QU-resolution *refutations, but admit polynomial-sized* $\mathsf{QCDCL}^{\textsf{Lev-Ord}}_{\textsf{All-Red,Exi-Prop}}$ *refutations.*

We combine the $\texttt{MirrorCR}$ and $\texttt{QParity}$ formulas into a new one, using auxiliary variables.

▶ **Definition 15.** *The QBF* $\texttt{MiPa}_n$ *consists of the prefix* $\forall z \exists X \forall u \exists T \forall p \exists Y \forall w \exists S \forall v \exists r$ *such that* X*,* u*,* T *are the variables for* $\texttt{MirrorCR}_n(X, u, T)$*, and* Y*,* w*,* S *are the variables for* $\texttt{QParity}_n(Y, w, S)$*. The matrix of* $\texttt{MiPa}_n$ *contains the clauses*

$$z \vee \bar{r}, \quad \bar{z} \vee \bar{r}$$

$$\left.\begin{array}{l} C \vee p \vee v \vee r \\ C \vee p \vee \bar{v} \vee r \\ C \vee \bar{p} \vee v \vee r \\ C \vee \bar{p} \vee \bar{v} \vee r \end{array}\right\} \textit{for } C \in \mathfrak{C}(\texttt{MirrorCR}_n(X, U, T)),$$

$$\left.\begin{array}{l} p \vee D \\ \bar{p} \vee D \end{array}\right\} \textit{for } D \in \mathfrak{C}(\texttt{QParity}_n(Y, w, S)).$$

We show next that $\texttt{MiPa}_n$ needs indeed ANY-RED in order to admit polynomial-size refutations in QCDCL. The idea is that ALL-RED will always lead to refutations of $\texttt{MirrorCR}_n$, and NO-RED will alternatively lead to Q-resolution refutations of $\texttt{QParity}_n$, which are both of exponential size.

▶ **Theorem 16.** *The QBFs* $\texttt{MiPa}_n$

(i) *need exponential-size* $\mathsf{QCDCL}^{\textsf{Lev-Ord}}_{\textsf{All-Red,Exi-Prop}}$ *refutations,*
(ii) *need exponential-size* $\mathsf{QCDCL}^{\textsf{Lev-Ord}}_{\textsf{No-Red,Exi-Prop}}$ *refutations,*
(iii) *but have polynomial-size* $\mathsf{QCDCL}^{\textsf{Lev-Ord}}_{\textsf{Any-Red,Exi-Prop}}$ *refutations.*

Proof. For (i), since the formula has no unit clauses, we have to start by deciding the variable z. Because z occurs symmetrically in $\texttt{MiPa}_n$, we can assume that we set z to true. This always triggers the unit propagation of $\bar{r}$ via the clause $\bar{z} \vee \bar{r}$. After that, we are forced to assign the variables from X, $U := \{u\}$ and T along the quantification order. Since the matrix of $\texttt{MirrorCR}_n$ is unsatisfiable, and we need to reduce all literals if possible, we will detect a conflict at the same time as we would get the conflict in $\texttt{MirrorCR}_n$ itself. The proof we can extract from the trails is essentially a $\mathsf{QCDCL}^{\textsf{Lev-Ord}}_{\textsf{All-Red,Exi-Prop}}$ refutation of $\texttt{MirrorCR}_n$, except that it additionally contains the variables z, p, v and r in some polarities. However, this does not change the fact that we can still not resolve two clauses that contain X-, U-, and T-variables over any X-variable. Therefore, if we shorten the proof by assigning r to false and z to true, we get a refutation of $\texttt{MirrorCR}_n$, in which we never resolve two clauses that contain X-, U-, and T-variables over an X-variable. This property is called *primitive* (cf. [16]). Also in [16], it was shown that primitive Q-resolution refutations of $\texttt{MirrorCR}_n$ need exponential size.

For (ii), we start in the same way as in (i), but we do not get a conflict once we assigned all variables of $\texttt{MirrorCR}_n$. Next, we need to decide p in some polarity, but nothing will happen for the moment. We then start assigning the variables of $\texttt{QParity}_n$ along the quantification order. Now we have to distinguish two cases:

Case 1: We get a conflict in $\texttt{QParity}_n$. But then, because of NO-RED, we can only extract Q-resolution derivations of learned clauses. And if we get enough conflicts in $\texttt{QParity}_n$, we can essentially extract a Q-resolution refutation of $\texttt{QParity}_n$, which has exponential size.

Case 2: We do not get a conflict in $\texttt{QPartity}_n$. This might happen when the universal player assigns the variable w the "wrong" way. Then the only unassigned variable is v. After deciding it in any polarity, we will always get a conflict in $\texttt{MirrorCR}_n$. If we find enough conflicts in $\texttt{MirrorCR}_n$, we can essentially extract an exponential-size fully reduced primitive Q-resolution refutation of $\texttt{MirrorCR}_n$ as in (i).

Note that it is possible to get both kind of conflicts. However, it is only important with what kind of conflicts we were able to derive the empty clause.

Finally, for (iii), we can construct a polynomial-size $\mathsf{QCDCL}^{\textsc{Lev-Ord}}_{\textsc{Any-Red,Exi-Prop}}$ proof by only reducing the literals w and $\bar{w}$. After deciding z, propagating $\bar{r}$, assigning all variables from X, u and T and deciding p arbitrarily, we can simply copy the polynomial-size $\mathsf{QCDCL}^{\textsc{Lev-Ord}}_{\textsc{All-Red,Exi-Prop}}$ proof of $\texttt{QParity}_n$ (note that ALL-RED only applies to w and $\bar{w}$). At some point, we will derive the clause (p) or $(\bar{p})$, which can be reduced to the empty clause. ◀

One of the initial motivations of this paper was to find a way to p-simulate long-distance Q-resolution refutations of QCNFs by certain variants of QCDCL. However, it appears that not all resolution steps that are allowed in long-distance Q-resolution can be recreated with QCDCL proofs. In long-distance Q-resolution proofs that are extracted from QCDCL, one can easily observe that for each resolution step $C_1 \overset{\ell}{\bowtie} C_2$, at least one parent clause C_i has to be an antecedent clause for ℓ or $\bar{\ell}$ in the corresponding trail. In particular, there must be a partial assignment τ and a set of literals K such that $\text{red}_K(C_i|_\tau)$ becomes unit, i.e. $\text{red}_K(C_i|_\tau) = (\ell)$ (resp. $(\bar{\ell})$). This is not possible if there are tautologies left of ℓ in C_i that cannot be reduced.

Motivated by this observation, we introduce a new proof system similar to long-distance Q-resolution, but with the restriction that such a situation as described above is not allowed.

▶ **Definition 17.** *A* long-distance Q-resolution *proof is called a* mLD-Q-Res proof*, if it does not contain a resolution step between two clauses D and E, such that $C = D \overset{x}{\bowtie} E$ for an existential variable x and there are universal variables u, w such that $u^* \in D$, $w^* \in E$ and $lv_\Phi(u), lv_\Phi(w) < lv_\Phi(x)$.*

With this definition in place, we can show that mLD-Q-Res proofs can be extracted from runs of most variants of QCDCL that we defined. Further, for some QCDCL paradigms, stricter simulations hold.

▶ **Proposition 18.** *The following holds on false QCNFs:*

(i) Q-resolution *p-simulates* $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{No-Red,Exi-Prop}}$.

(ii) QU-resolution *p-simulates* $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{No-Red,All-Prop}}$.

(iii) mLD-Q-Res *p-simulates* $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{Any-Red,Exi-Prop}}$.

Proof. Item (i) was already shown in [10].

For (ii), because of ALL-PROP, we might propagate (and resolve) over universal literals, which can be handled by QU-resolution. It remains to show that NO-RED prevents the derivation of tautological clauses. This holds because we only use antecedent clauses for clause learning. Let us assume we learn a tautological clause C from a $\mathsf{QCDCL}^{\text{ANY-ORD}}_{\text{NO-RED,ALL-PROP}}$ trail $\mathcal{T}$. Then there would be two antecedent clauses $D := \text{ante}_{\mathcal{T}}(\ell_1)$ and $E := \text{ante}_{\mathcal{T}}(\ell_2)$ such that there exists a universal literal u with $u \neq \ell_1$, $\bar{u} \neq \ell_2$, $u \in D$ and $\bar{u} \in E$. We need $\bar{u} \in \mathcal{T}$ for D to become unit and at the same time we need $u \in \mathcal{T}$ for E to become unit, which is not possible. Therefore, we will never derive tautological clauses.

Let us now prove (iii). By definition, we can extract long-distance Q-resolution proof from $\mathsf{QCDCL}^{\text{ANY-ORD}}_{\text{ANY-RED,EXI-PROP}}$ trails (note that we only propagate existential literals, hence we also only resolve over existential variables during clause learning). It remains to show that the kind of resolution step that is forbidden in mLD-Q-Res (but allowed in long-distance Q-resolution) will never occur during clause learning.

Assume it does. Then we have derived a clause C by resolving two clauses D and E over some literal x (hence $C = D \overset{x}{\bowtie} E$), such that there exists universal tautologies $u^* \in D$ and $w^* \in E$ with $u^* \neq w^*$ and $\text{lv}(u^*), \text{lv}(w^*) < \text{lv}(x)$. Then at least one of these parent clauses needs to be an antecedent clause for a trail $\mathcal{T}$, say $D = \text{ante}_{\mathcal{T}}(x)$. But then D can never become the unit clause (x), because we cannot reduce u^* since it is blocked by x, and we cannot falsify it by the previous trail assignment since it is a tautology. This is a contradiction that shows that all resolution and reduction steps are allowed in mLD-Q-Res. ◀

We could formulate analogous results on true QCNFs using the notation of consensus proofs. However, we will omit this as all separations and characterisations will be performed on false QCNFs and resolution proofs.

One can easily show that the separation between Q-resolution and long-distance Q-resolution transfers to a separation between Q-resolution and mLD-Q-Res.

▶ **Corollary 19.** mLD-Q-Res *p-simulates and is exponentially stronger than* Q-resolution*.*

Proof. The simulation follows by definition. The separation follows by Theorem 14 and Proposition 18 (iii). ◀

In fact, all currently known upper bounds for long-distance Q-resolution can be easily transformed into mLD-Q-Res upper bounds. However, we leave open the question whether long-distance Q-resolution is stronger than or equivalent to mLD-Q-Res.

5 Characterisations of QU-resolution and mLD-Q-Res

In this section, show that all the simulations in Proposition 18 can be tightened to equivalences. For this we will characterise both mLD-Q-Res and QU-resolution by the specific variants of QCDCL mentioned in Proposition 18. Characterising Q-resolution by $\mathsf{QCDCL}^{\text{ANY-ORD}}_{\text{NO-RED,EXI-PROP}}$ was already undertaken in [10]. However, we leave open, whether we can extend these characterisations to long-distance Q-resolution. This will depend on whether it is possible to polynomially transform the "forbidden" resolution steps that can occur in long-distance Q-resolution, but cannot be created by QCDCL, into mLD-Q-Res steps.

The characterisations follow the same idea as in [10], in which Q-resolution was characterised. One crucial difference is that we now want to use the ANY-RED policy, i.e., in each step we have to decide what literals to reduce.

As already mentioned in Remark 5, it suffices to update the reductive sets only after a conflict. That means that for characterising mLD-Q-Res, it is enough to fix the literals that are going to be reduced throughout the whole trail. Thus, we introduce the notion of *L-reductive trails.*

▶ **Definition 20** (L-reductive trails)**.** *Let L be a set of literals. A trail $\mathcal{T}$ is called L-*reductive*, if for each propagation step in $\mathcal{T}$ the literals that were selected to be reduced are exactly the literals in L. Formally, this means that for each $p_{(i,j)}$ there is an antecedent clause (resp. cube) $ante_{\mathcal{T}}(p_{(i,j)})$ such that $red_L(ante_{\mathcal{T}}(p_{(i,j)})|_{\mathcal{T}[i,j]}) = (p_{(i,j)})$ (resp. $[\bar{p}_{(i,j)}]$).*

Before starting with a new L-reductive trail, we always need to consider the choice of the reductive set L. As we know from [10] and Proposition 18, tautologies can only be created when the corresponding literal got reduced somewhere in the trail. In fact, since $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{No-Red,Exi-Prop}}$ already characterises Q-resolution [10], we can conclude that in some sense the only purpose of reductions during unit propagation is to create tautological clauses. Therefore we will distinguish between the tautological and the non-tautological part of a clause.

▶ **Definition 21.** *Let C be a clause. Let $G(C) := \{u \in C : u$ is universal and $\bar{u} \in C\}$. This set is the* tautological part *of C. The* non-tautological part *$H(C)$ of C is defined as $H(C) := C \backslash G(C)$.*

For each QU-resolution proof π and $C \in \pi$ we have $G(C) = \varnothing$.

Our next notion is similar to the concepts of *unreliable* [10] and *1-empowering* [26].

▶ **Definition 22** (Blockades)**.** *Let $\mathsf{S} \in \{\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{Any-Red,Exi-Prop}}, \mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{No-Red,All-Prop}}\}$ and C be a clause. A tuple $(\mathcal{U}, \alpha, \ell, K)$, where $\mathcal{U}$ is a trail, ℓ is a literal, α is a non-tautological set of literals and K is a set of universal literals, is called a* blockade of C with respect to S *for a QCNF $\Phi = \mathcal{Q} \cdot \phi$, if $\mathcal{U}$ is a K-reductive S trail with decisions α, such that $\ell \in C$, $\alpha \subseteq \overline{C} \backslash \{\bar{\ell}\}$, $K \subseteq G(C)$ and $\alpha \cap K = \varnothing$.*

For $\mathsf{S} = \mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{Any-Red,Exi-Prop}}$, we additionally require that ℓ is an existential literal and α consists of only existential literals.

▶ **Example 23.** Blockades occur when we are not able to choose all decisions from a pre-defined non-tautological set α. For example, consider the QCNF

$$\exists x, y \forall u, v \exists z \; (\bar{y} \vee \bar{z}) \wedge (\bar{x} \vee \bar{u} \vee z) \wedge (x \vee y \vee v \vee z) \wedge (y \vee \bar{v} \vee z).$$

Assume that we use $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{Any-Red,All-Prop}}$. Then the clause $C := \bar{x} \vee \bar{y} \vee u \vee \bar{u} \vee z$ has a blockade $(\mathcal{U}, \alpha, \ell, K)$ with $\mathcal{U} := (\mathbf{y}, \bar{z}, \bar{x})$, where $\mathrm{ante}_{\mathcal{U}}(\bar{z}) = \bar{y} \vee \bar{z}$, $\mathrm{ante}_{\mathcal{U}}(\bar{x}) = \bar{x} \vee \bar{u} \vee z$, as well as $\ell := \bar{x} \in C$, $\alpha := \{y\} \subseteq \overline{C} \backslash \{\bar{\ell}\}$ and $K := \{\bar{u}\}$.

Intuitively, this means that although the clause C is not directly contained in the formula, we are still able to detect the implication $(\alpha \wedge \overline{K} \rightarrow \ell) = (y \wedge u) \rightarrow \bar{x}$ (which is equivalent to $\bar{y} \vee \bar{u} \vee \bar{x} \subseteq C$) as a composition of decisions and unit propagations. It turns out that, instead of learning C directly, it is enough to detect a blockade in order to make use of C for unit propagations in later trails.

The next lemma shows, that we can recall trails (and blockades in particular), that were detected and stored at an earlier point, and restore all propagations they contained. This will be important for the characterisations, as we will go through the given proof, find blockades or conflicts for all clauses in that proof, and recall the corresponding trails (by using this Lemma) an all their containing propagations whenever the clauses are needed for another resolution step. In that way, we can virtually store previous trails and recall them later again as natural trails.

▶ **Lemma 24.** *Let $\Phi = \mathcal{Q} \cdot \phi$ and $\Psi = \mathcal{Q} \cdot \psi$ be QCNFs such that $\psi \subseteq \phi$.*

Let $\mathcal{U}$ be a K-reductive trail (for NO-RED *we set $K = \emptyset$) for the QCNF Ψ with decisions β. Let $\mathcal{T}$ be a natural L-reductive trail ($L = \emptyset$ for* NO-RED*) with decisions α for the QCNF Φ such that $K \subseteq L$, $\beta \subseteq \mathcal{T}$ and $\alpha \cap L = \emptyset$. If $\mathcal{T}$ does not run into a clause conflict, then all propagated literals from $\mathcal{U}$ are also contained in $\mathcal{T}$.*

Proof. Assume that $\mathcal{T}$ does not run into a clause conflict, but there are some propagated literals from $\mathcal{U}$ that are not contained in $\mathcal{T}$. Let $p_{(a,b)}$ be the literal that is leftmost in $\mathcal{U}$ with this property and define $A := \text{ante}_{\mathcal{U}}(p_{(a,b)})$. Since there are no cubes present, we conclude that A must be a clause, regardless of whether $p_{(a,b)}$ is existential or universal.

Because $p_{(a,b)}$ is leftmost, all other propagated literals before $p_{(a,b)}$ in $\mathcal{U}$ are already contained in $\mathcal{T}$. Since $\mathcal{U}$ was K-reductive, we know that $\text{red}_K(A|_{\mathcal{U}[a,b]}) = (p_{(a,b)})$. Because of $K \subseteq L$ and $\mathcal{U}[a,b] \subseteq \mathcal{T}$ we have either $\text{red}_L(A|_{\mathcal{T}}) \in \{(p_{(a,b)}), (\bot)\}$, or $A|_{\mathcal{T}}$ becomes true. Note that we can set $K := L := \emptyset$ for the rest of our argumentation in the case where $p_{(a,b)}$ is universal.

The first case would contradict our assumption (since $\mathcal{T}$ is natural), therefore we have to assume that $A|_{\mathcal{T}}$ becomes true. This means that we can find a literal $p_{(a,b)} \neq u \in A \cap \mathcal{T}$. If u was existential, then we would need $\bar{u} \in \mathcal{U}[a,b]$. But this would also imply $\bar{u} \in \mathcal{T}$ which contradicts the fact that $u \in \mathcal{T}$. Hence u must be universal.

If u was a decision in $\mathcal{T}$, then we would have $u \in \alpha$. Because of $\alpha \cap L = \emptyset$ we conclude $u \notin L$ and also $u \notin K$. In order to make u vanish in $\text{red}_K(A|_{\mathcal{U}[a,b]})$, we need $\bar{u} \in \mathcal{U}[a,b]$, hence also $\bar{u} \in \mathcal{T}$. However, this is a contradiction because we already assumed $u \in \mathcal{T}$.

Therefore, u must have been propagated by an antecedent clause $\text{ante}_{\mathcal{T}}(u)$. But then we have $K = \emptyset$, hence $u \notin K$ and $\bar{u} \in \mathcal{U}[a,b] \subseteq \mathcal{T}$, which is a contradiction again because of $u \in \mathcal{T}$. ◀

▶ **Theorem 25.** *The following holds:*
- $\mathsf{QCDCL}^{\textsf{Any-Ord}}_{\textsf{Any-Red,Exi-Prop}}$ *p-simulates* mLD-Q-Res.
- $\mathsf{QCDCL}^{\textsf{Any-Ord}}_{\textsf{No-Red,All-Prop}}$ *p-simulates* QU-resolution.

In detail: Let $\Phi = \mathcal{Q} \cdot \phi$ be a QCNF in n variables and $\pi = D_1, \dots, D_m$ be a mLD-Q-Res *(*QU-resolution*) refutation of Φ. Then we can construct a $\mathsf{QCDCL}^{\textsf{Any-Ord}}_{\textsf{Any-Red,Exi-Prop}}$ ($\mathsf{QCDCL}^{\textsf{Any-Ord}}_{\textsf{No-Red,All-Prop}}$) refutation ι of Φ with $|\iota| \in \mathcal{O}(n \cdot |\pi|)$.*

Proof. We only sketch the proof here.

Going through a given mLD-Q-Res (QU-resolution) refutation π, starting at the axioms, for each $C \in \pi$ we create specific natural trails (where some of them will later be part of the $\mathsf{QCDCL}^{\textsf{Any-Ord}}_{\textsf{Any-Red,Exi-Prop}}$ or $\mathsf{QCDCL}^{\textsf{Any-Ord}}_{\textsf{No-Red,All-Prop}}$ proof) in which all decisions are negated literals from C, until one of the following events occur:
- We get a conflict and learn a subclause of C.
- We obtain a blockade of C.

When this happens, we either assign the label "subclause" or the label "blockade" to C. When a clause was derived via a resolution or reduction step in π, we simply recall the blockades of its parent clauses by applying Lemma 24 to create a blockade for the resolvent or a conflict. If a parent clause does not have a blockade, the clause itself (or a subclause) must have been learned directly and can therefore be used as an antecedent clause for the trail that either becomes a blockade for the resolvent, or that runs into a conflict from which we can learn a subclause of the resolvent.

Since a clause $C \in \pi$ can be derived via resolution (say $C = D \bowtie E$) or reduction (say $C = \text{red}(D)$), we have to consider all possible cases:

- resolution, both D and E are labelled "blockade"
- resolution, D is labelled "blockade", E is labelled "subclause", or vice versa
- resolution, both D and E are labelled "subclause"
- reduction, D is labelled "blockade"
- reduction, D is labelled "subclause"

At the end, each clause in π is either labelled "subclause" or "blockade". In particular, this holds for the empty clause. Because, by definition, there cannot be a blockade of the empty clause (we need at least one literal), the empty clause must be labelled "subclause", which means we have learned the empty clause. ◀

Proposition 18 and Theorem 25 yield the following characterisations:

▶ **Corollary 26.** $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{Any-Red},\textsc{Exi-Prop}} \equiv_p \mathsf{mLD\text{-}Q\text{-}Res}$ *and* $\mathsf{QCDCL}^{\textsc{Any-Ord}}_{\textsc{No-Red},\textsc{All-Prop}} \equiv_p \mathsf{QU\text{-}Res}$.

▶ Remark 27. Note that our simulations require a particular learning scheme, in which we almost always restart after each conflict. This is also the reason why we get an improved simulation complexity of $\mathcal{O}(n \cdot |\pi|)$ compared to $\mathcal{O}(n^3 \cdot |\pi|)$ from [10], in which arbitrary (asserting) learning schemes were allowed (where we do not necessarily restart every time).

Performing our simulation under arbitrary asserting learning schemes might require some additional analysis on asserting clauses under the ANY-ORD and ANY-RED rules, as a clause learned from a K_1-reductive trail might not be asserting in K_2-reductive trails anymore. However, if it was clear how to guarantee asserting clauses in our systems, we would be able to obtain similar results as in [10], that is:

- For each clause C in the given mLD-Q-Res (QU-resolution) refutation and an arbitrary asserting learning scheme, we need $\mathcal{O}(n^2)$ trails and backtracking steps until we either learn a subclause of C, or we receive a blockade for C.
- Under any arbitrary asserting learning scheme, we can perform the simulation in time $\mathcal{O}(n^3 \cdot |\pi|)$. In particular, we do not need to restart after each conflict.

6 Conclusion

Proving theoretical characterisations of QCDCL variants successfully used in practice is an important and compelling endeavour. While we contributed to this line research, a number of open questions remain, both theoretically and practically. In particular, in light of Figure 1, it seems worthwhile to explore whether some of the QCDCL models shown to be theoretically better than standard QCDCL can be used for practical solving.

In our quest to modify QCDCL to match the strength of its underlying system long-distance Q-resolution, we introduced the new proof system mLD-Q-Res, which not only characterises a strong version of QCDCL, but also simulates all related variants. This allows to use proof-theoretic results for mLD-Q-Res whenever considering the strength of QCDCL solvers. Yet, we leave open whether mLD-Q-Res is strictly weaker than or equivalent to long-distance Q-resolution. Both possible outcomes would be interesting, as either long-distance Q-resolution does not characterise QCDCL, or there are modifications of QCDCL that unleash the full strength of long-distance Q-resolution.

Additionally, we exhibited a QCDCL version characterising QU-resolution. One could try to combine these two characterisations to obtain an even stronger family of QCDCL variants in the spirit of LDQU$^+$-resolution. Further, cube learning, which can hugely impact the running time even on false formulas [17], was not considered here. Hence, verifying true formulas as well as the proof-theoretic characterisation of modifications to QCDCL such as dependency learning [25] are further topics for future research.

References

1 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011.

2 Albert Atserias and Moritz Müller. Automating resolution is NP-hard. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 498–509. IEEE Computer Society, 2019.

3 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Form. Methods Syst. Des.*, 41(1):45–65, 2012.

4 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *Proc. Theory and Applications of Satisfiability Testing (SAT)*, pages 154–169, 2014.

5 Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res. (JAIR)*, 22:319–351, 2004. `doi:10.1613/jair.1410`.

6 Olaf Beyersdorff. Proof complexity of quantified Boolean logic – a survey. In Marco Benini, Olaf Beyersdorff, Michael Rathjen, and Peter Schuster, editors, *Mathematics for Computation (M4C)*, pages 353–391. World Scientific, 2022.

7 Olaf Beyersdorff, Joshua Blinkhorn, and Luke Hinde. Size, cost, and capacity: A semantic technique for hard random QBFs. In *Proc. Conference on Innovations in Theoretical Computer Science (ITCS'18)*, pages 9:1–9:18, 2018.

8 Olaf Beyersdorff, Joshua Blinkhorn, and Meena Mahajan. Building strategies into QBF proofs. *J. Autom. Reason.*, 65(1):125–154, 2021. `doi:10.1007/s10817-020-09560-1`.

9 Olaf Beyersdorff, Joshua Blinkhorn, Meena Mahajan, and Tomás Peitl. Hardness characterisations and size-width lower bounds for QBF resolution. *ACM Transactions on Computational Logic*, 2022.

10 Olaf Beyersdorff and Benjamin Böhm. Understanding the Relative Strength of QBF CDCL Solvers and QBF Resolution. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:20, 2021.

11 Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Jan Pich. Frege systems for quantified Boolean logic. *J. ACM*, 67(2):9:1–9:36, 2020.

12 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. New resolution-based QBF calculi and their proof complexity. *ACM Transactions on Computation Theory*, 11(4):26:1–26:42, 2019.

13 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof complexity of resolution-based QBF calculi. In *Proc. Symposium on Theoretical Aspects of Computer Science (STACS'15)*, pages 76–89. LIPIcs, 2015.

14 Olaf Beyersdorff, Mikolás Janota, Florian Lonsing, and Martina Seidl. Quantified Boolean formulas. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, pages 1177–1221. IOS Press, 2021.

15 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.

16 Benjamin Böhm and Olaf Beyersdorff. Lower bounds for QCDCL via formula gauge. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing (SAT)*, pages 47–63, Cham, 2021. Springer International Publishing.

17 Benjamin Böhm, Tomás Peitl, and Olaf Beyersdorff. QCDCL with cube learning or pure literal elimination – What is best? In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1781–1787. ijcai.org, 2022.

18 Benjamin Böhm, Tomás Peitl, and Olaf Beyersdorff. Should decisions in QCDCL follow prefix order? In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 236 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

19 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, pages 233–350. IOS Press, 2021.

20 Mikolás Janota. On Q-Resolution and CDCL QBF solving. In *Proc. International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 402–418, 2016.

21 Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015.

22 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

23 Jan Krajíček. *Proof complexity*, volume 170 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 2019.

24 João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2021.

25 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Dependency learning for QBF. *J. Artif. Intell. Res.*, 65:180–208, 2019.

26 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. doi:10.1016/j.artint.2010.10.002.

27 Friedrich Slivovsky. Quantified CDCL with universal resolution. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 236 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

28 Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In *Proc. Principles and Practice of Constraint Programming (CP)*, pages 647–663, 2012.

29 Moshe Y. Vardi. Boolean satisfiability: theory and engineering. *Commun. ACM*, 57(3):5, 2014.

30 Marc Vinyals. Hard examples for common variable decision heuristics. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

31 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *Proc. IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, pages 442–449, 2002.

Polynomial Calculus for MaxSAT

Ilario Bonacina
Polytechnic University of Catalonia, Barcelona, Spain

Maria Luisa Bonet
Polytechnic University of Catalonia, Barcelona, Spain

Jordi Levy
Artificial Intelligence Research Institute, Spanish Research Council (IIIA-CSIC), Barcelona, Spain

Abstract

MaxSAT is the problem of finding an assignment satisfying the maximum number of clauses in a CNF formula. We consider a natural generalization of this problem to generic sets of polynomials and propose a weighted version of Polynomial Calculus to address this problem.

Weighted Polynomial Calculus is a natural generalization of MaxSAT-Resolution and weighted Resolution that manipulates polynomials with coefficients in a finite field and either weights in $\mathbb{N}$ or $\mathbb{Z}$. We show the soundness and completeness of these systems via an algorithmic procedure.

Weighted Polynomial Calculus, with weights in $\mathbb{N}$ and coefficients in $\mathbb{F}_2$, is able to prove efficiently that Tseitin formulas on a connected graph are minimally unsatisfiable. Using weights in $\mathbb{Z}$, it also proves efficiently that the Pigeonhole Principle is minimally unsatisfiable.

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Automated reasoning

Keywords and phrases Polynomial Calculus, MaxSAT, Proof systems, Algebraic reasoning

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.5

Funding This work was supported by the Spanish Agencia Estatal de Investigación with the grant PID2019-109137GB-C21/AEI/10.13039/501100011033 and the grant PID2019-109137GB-C22/AEI/10.13039/501100011033.

Ilario Bonacina: Partially supported by the grant IJC2018-035334-I/AEI/10.13039/501100011033.

Maria Luisa Bonet: Partially supported by Simons Institute.

Jordi Levy: Partially supported by Simons Institute.

1 Introduction

The question of whether a set of polynomials $F = \{f_1, \ldots, f_m\}$ is satisfiable – i.e. to know if there exists an assignment of the variables α s.t. $f_i(\alpha) = 0$ for every i – is at the root of algebraic geometry, and it is a natural generalization of SAT, since we can encode CNF formulas as sets of polynomials (over $\{0, 1\}$-valued variables).

The state-of-the-art of practical SAT solving is dominated by CDCL SAT solvers, all of them based on the Resolution proof system [28, 3]. In the last two decades, these solvers have reached remarkable efficiency in industrial SAT instances, but to get further substantial improvements we think it will be necessary to broaden the current paradigm beyond Resolution. Therefore it makes sense to look at the problem from a different point of view using algebraic language and methods to have an impact on solving instances.

Another line of investigation is focusing on encodings to overcome the limitations of CDCL solving. For instance, [21, 5] has shown that the dual-rail encoding allows translating SAT instances into MaxSAT problems. This results in translations of the Pigeonhole Principle with polynomial size proofs using MaxSAT resolution. The same applies to the translation of SAT to Max2SAT using the gadget described in [2]. Moreover, in both cases, general-purpose MaxSAT solvers are able to solve these instances in practice, even though these MaxSAT solvers are not based on MaxSAT resolution.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 5; pp. 5:1–5:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We have algebraic systems that are stronger than Resolution, and therefore it makes sense to extend those systems to solve MaxSAT problems to see if we can improve on the dual-rail and Max2SAT translations. Moreover, algebraic systems inherently allow more alternative encodings of the problems. For instance, we can use a direct encoding into polynomials, or encode first via a CNF and then translate them into polynomials. These encodings allow us to use algorithms to compute Groebner bases [10, 9, 18], and efficiency maybe gained by these changes. For instance, a direct algebraic encoding, and Groebner-based techniques are useful in practice for coloring [14, 15, 16] and the verification of multiplier circuits [25, 23, 24, 22]. The proof system capturing Groebner-based algorithms is *Polynomial Calculus* (PC) [12], which is a proof system strictly stronger than Resolution. Polynomial Calculus is degree-automatable, in the sense that bounded degree proofs can be found efficiently (in time $n^{O(d)}$, where d is the degree). This is one more reason to extend PC techniques to MaxSAT.

In this paper, we consider the generalization of MaxSAT to the context of polynomials, that is the question of what is the maximum number of polynomials of a given set we are able to simultaneously satisfy. Equivalently, the minimum number of polynomials that we cannot satisfy. In this algebraic context, there are also MaxSAT problems that have natural direct encodings as sets of polynomials, for instance, max-cut or max-coloring.

We define an extension of PC suitable for MaxSAT, i.e. a system that not only is able to prove that a set of polynomials is unsatisfiable but to prove what is the maximum number of polynomials that can be satisfied simultaneously.

Our generalization of PC to a system suitable for MaxSAT is done in a similar way as the adaptation of Resolution to systems suitable for MaxSAT, for instance, MaxSAT-Resolution [7, 8], and weighted Resolution [6]. As weighted Resolution is a system for MaxSAT handling *weighted* clauses, we consider weighted PC, a system handling weighted polynomials. We consider polynomials over *finite* fields. The intuitive reason for this is that, over a finite field $\mathbb{F}_q$ with q elements, we can express $f \neq 0$ as the polynomial equality $f^{q-1} = 1$. We define weighted Polynomial Calculus for polynomials with coefficients in $\mathbb{F}_2$ in Section 3 and in Section 6 we give the definition in the general case.

We call $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ the weighted version of Polynomial Calculus handling weighted polynomials with coefficients in $\mathbb{F}_2$ and weights in $\mathbb{N}$. Intuitively the positive weight of a clause/polynomial is the "penalty" we pay to falsify it. Weighted Resolution has been generalized to $\mathbb{Z}$-weighted Resolution, i.e. weighted resolution but with negative weights [27, 6, 26, 30]. In a similar way, we also define $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ as $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ but where we allow negative weights in the proofs. Intuitively the meaning of a weighted clause/polynomial with a negative weight is that it is introduced in the proof as an "assumption" to be justified later, and the negative weight is to keep track of such assumptions yet to be justified.

Connections of weighted Polynomial Calculus with other MaxSAT systems

It is well known that PC (with an appropriate encoding of CNF formulas, the twin variable encoding, see Section 2.2) simulates Resolution. This immediately gives that $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ with twin variables is as strong as $\mathbb{N}$-weighted Resolution and $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ is as strong as $\mathbb{Z}$-weighted Resolution (aka Sherali-Adams and Circular Resolution [4, 6]). Pictorially the relations between $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}/\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ and the aforementioned systems can be summarized as in Fig. 1.

None of the systems above are equivalent since $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ and $\mathbb{Z}$-weighted Resolution are incomparable. In one direction Tseitin(G) is easy in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ while it is hard for $\mathbb{Z}$-weighted Resolution. This follows from the lower bound in [20]. (Tseitin formulas are treated in more detail in Section 5.) In the other direction, the Pigeonhole Principle is easy for $\mathbb{Z}$-weighted

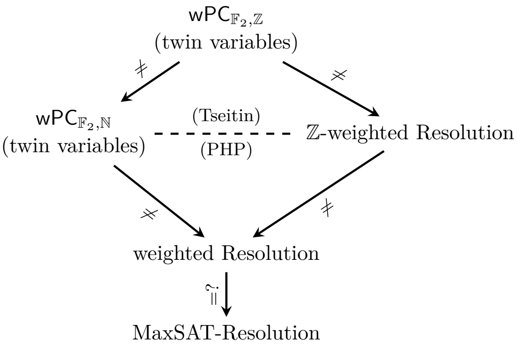


Figure 1 $P \to Q$ means that P is at least as strong as Q. A dashed line means the two systems are incomparable.

Resolution (see for instance [4]) and hence for $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$, while it is hard for $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$. This follows from the lower bound on the Pigeonhole principle in Polynomial Calculus [29]. Both the Pigeonhole principle and Tseitin have short proofs in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$.

We recall that Figure 1 can also be read in the context of propositional proof systems. Indeed, all the MaxSAT systems in Figure 1 can be seen as propositional proof systems, if the weights of the initial clauses/polynomials are *not* part of the input but part of the proof and to refute we just want to derive one instance of the empty clause or the polynomial 1. In this setting weighted Resolution is the same as Resolution and $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ is the same as PC over $\mathbb{F}_2$.

Analogous simulations as the ones in Fig. 1 hold also for $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{N}}/\mathsf{wPC}_{\mathbb{F}_q,\mathbb{Z}}$.

The main technical contribution of this work is the proof of the completeness of $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{N}}$. This is proved in detail for $\mathbb{F}_2$ in Section 4 and we show how to adapt it to the general case in Section 6. The completeness is proved via a saturation process which is a natural generalization of an analogous process used in [7, 8, 1] to prove the completeness of MaxSAT-Resolution. Unlike for MaxSAT-Resolution, we take a more semantic view of the process which makes easier to adapt it to the context of polynomials.

Structure of the paper

Section 2 contains all the necessary preliminaries, in particular, the definition of PC and the extension of MaxSAT to polynomials. In Section 3, we give the formal definition of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ and $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$. Section 4 contains the completeness of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$. In Section 5, we give an application of the saturation process to Tseitin formulas. Section 6 shows how to generalize the definition of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ and $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ from $\mathbb{F}_2$ to a generic finite field. Finally, in Section 7, we give some concluding remarks.

2 Preliminaries

For $n \in \mathbb{N}$, let $[n] = \{1, \dots, n\}$. In general, we use capital letters to denote (multi-)sets.

2.1 Polynomial Calculus

Let $\mathbb{F}_q$ be a finite field with q elements (it exists whenever $q = p^k$ for some prime p and integer k). For most of this paper, we focus on $q = 2$, i.e. on the field with two elements $0, 1$. Given a set of variables X, with $\mathbb{F}_q[X]$ we denote the ring of multivariate polynomials with coefficients in $\mathbb{F}_q$ and variables in X.

We denote polynomials using the letters f, g, while we use Greek letters to denote assignments. An assignment is a function $\alpha\colon X \to \mathbb{F}_q$ and for a polynomial $f \in \mathbb{F}_q[X]$, $f(\alpha)$ is the *evaluation* of f in α: the element of $\mathbb{F}_q$ resulting from substituting each variable x in f with $\alpha(x)$ and simplifying the resulting expression. If $f(\alpha) = 0$ we say that α *satisfies* f. The polynomial 1 represents the unsatisfiable polynomial (the equivalent to the empty clause in SAT).

Next, we define the algebraic proof system *Polynomial Calculus*, originally introduced by Clegg et al. [12]. Even though the system can be defined for any field (or even rings, see for instance [11]), in this paper we only consider finite fields.

Polynomial Calculus over $\mathbb{F}_q$ ($\mathsf{PC}_{\mathbb{F}_q}$) is a proof system that handles polynomials in $\mathbb{F}_q[X]$. A derivation in $\mathsf{PC}_{\mathbb{F}_q}$ of a polynomial f from a set of polynomials F is a sequence of polynomials $f_1, \dots, f_m$, where $f_m = f$, and for each i either $f_i \in F$, or $f_i = gf_j$ for some $g \in \mathbb{F}_q[X]$ and $j < i$, or $f_i = f_j + f_k$ for some $j, k < i$. A *refutation* is a derivation of the polynomial 1, and the *size* of a derivation is the total number of bits needed to express it.

Sometimes, the inference rules of $\mathsf{PC}_{\mathbb{F}_q}$ are given as

$$\frac{f \quad g}{f+g}\,, \qquad \frac{f}{\alpha f} \qquad \text{and} \qquad \frac{f}{xf}$$

for all $f, g \in \mathbb{F}_q[X]$, $\alpha \in \mathbb{F}_q$, and $x \in X$. This will just give a polynomial increase in the size of the derivations (hence it is p-equivalent to our presentation of $\mathsf{PC}_{\mathbb{F}_q}$). $\mathsf{PC}_{\mathbb{F}_q}$ – together with an encoding of formulas into polynomials – is a Cook-Reckhow propositional proof system [13].

2.2 From formulas in CNF to polynomials

A *clause* C is a set of literals, i.e. Boolean variables x_i or negated Boolean variables $\neg x_i$ from a given set of variables $\{x_1, \dots, x_n\}$. A CNF formula is a set of clauses, and a k-CNF is a CNF where each clause has at most k literals. An assignment $\alpha\colon \{x_1, \dots, x_n\} \to \{0, 1\}$ satisfies a clause if it maps at least a literal to 1, where $\alpha(\neg x_i) := 1 - \alpha(x_i)$. An assignment satisfies a CNF formula if it satisfies all the clauses in it.

To encode CNF formulas into sets of polynomials that could be refuted in $\mathsf{PC}_{\mathbb{F}_q}$, we use the following encoding with *twin variables.* We call *twin variables* the set of variables $X = \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$. The intended meaning of $\bar{x}_i$ is $1 - x_i$.

For every clause $C = \{x_i : i \in I\} \cup \{\neg x_j : j \in J\}$, we associate the monomial $M(C) = \prod_{i \in I} \bar{x}_i \prod_{j \in J} x_j$ in the twin variables X. Then a set of clauses $\{C_1, \dots, C_m\}$ is encoded as

$$\{M(C_1),\ \dots,\ M(C_m)\} \cup \{x_i^2 - x_i,\ x_i + \bar{x}_i - 1\ \ : i \in [n]\}\ .$$

Any assignment $\alpha\colon \{x_1, \dots, x_n\} \to \{0, 1\}$ can be extended to an assignment $\alpha' : X \to \{0, 1\}$, where for each $i \in [n]$, $\alpha'(x_i) = \alpha(x_i)$ and $\alpha'(\bar{x}_i) = 1 - \alpha(x_i)$. Then α satisfies a CNF formula (i.e. α maps all the clauses to 1) if and only if α' satisfies the polynomial encoding of F (i.e. α' is a common solution of the polynomials).

With this encoding of CNF formulas into polynomials, it is well-known that for every q, $\mathsf{PC}_{\mathbb{F}_q}$ p-simulates Resolution and indeed the p-simulation is strict [11]. For example, Tseitin formulas (see Section 5) have polynomial size $\mathsf{PC}_{\mathbb{F}_2}$ refutations while they require exponential size in Resolution [32]. Notice that the variables $\bar{x}_i$s are not strictly needed for the encoding (they could be eliminated just by substituting $1 - x_i$ for each occurrence of $\bar{x}_i$), but $\mathsf{PC}_{\mathbb{F}_q}$ with this alternative encoding does not p-simulate Resolution, not even on k-CNFs [17]. With different encodings of CNF formulas, for instance, using $\{\pm 1\}$-valued variables, it is open whether $\mathsf{PC}_{\mathbb{F}_q}$ simulates Resolution (see [31] for lower bounds on $\mathsf{PC}_{\mathbb{Q}}$ with the ± 1-variables).

2.3 MaxSAT on sets of polynomials

Let X be a generic set of variables. To define partial weighted MaxSAT, we distinguish between hard and soft clauses. The hard clauses need to be satisfied, while the soft ones consist of a clause and a weight (a number in $\mathbb{N}$). The weight of a soft clause is the cost of falsifying it. Given a set of soft clauses F and a set of hard ones H, *Weighted Partial MaxSAT* is the problem of finding an assignment to the variables X that satisfies the hard clauses H, and that minimizes the cost of the falsified soft clauses F.

In this paper, we generalize partial weighted MaxSAT to arbitrary polynomials in $\mathbb{F}_q[X]$. The generalization of the *hard* constraints of MaxSAT is some set of polynomials $H \subset \mathbb{F}_q[X]$, while the generalization of the *soft* constraints is a multi-set of the form

$$F = \{[\, f_1 \,,\, w_1 \,], \dots, [\, f_m \,,\, w_m \,]\} \ ,$$

where $f_i \in \mathbb{F}_q[X]$ and $w_i \in \mathbb{N}$. A pair $[\, f \,,\, w \,]$ where $f \in \mathbb{F}_q[X]$ and $w \in \mathbb{Z}$ is a *weighted polynomial.* The weight w informally measures the “importance” we give to satisfying the polynomial f. In this context, we are interested in assignments α that minimize the weight of the falsified soft polynomials in F, and satisfy all the polynomials in H.

▶ **Definition 2.1** (H-compatible assignment). *Let $H \subseteq \mathbb{F}_q[X]$. An assignment $\alpha\colon X \to \mathbb{F}_q$ is H-*compatible *if for every $h \in H$, $h(\alpha) = 0$.*

Now, for each assignment $\alpha\colon X \to \mathbb{F}_q$, we measure how close it is to satisfying all the polynomials in F, and we do this by defining its *cost* as the sum of the weights of the polynomials in F not satisfied by α. Therefore, the cost of the assignment α on F is

$$\mathrm{cost}(\alpha, F) = \sum_{i\in[m]} w_i \chi_i(\alpha) \ , \tag{1}$$

where $\chi_i(\alpha)$ is 1 if $f_i(\alpha) \neq 0$, and 0 otherwise. Finally, to solve a partial weighted MaxSAT problem, we want the minimum value of $\mathrm{cost}(\alpha, F)$ for any H-compatible assignment α, i.e.

$$\mathrm{cost}_H(F) = \min_{\alpha\ H\text{-compatible}} \mathrm{cost}(\alpha, F) \ . \tag{2}$$

If $H = \emptyset$, we denote $\mathrm{cost}_H(F)$ simply as $\mathrm{cost}(F)$. Of course, if F is satisfiable using a H-compatible assignment, then $\mathrm{cost}_H(F) = 0$.

Notice that, the polynomials in H and the weighted polynomials in F could come from the translation of a partial weighted MaxSAT instance. However, we cannot assume this is always the case. Moreover, the polynomials in H could be used to enforce specific types of assignments.

▶ **Example 2.2** (Boolean axioms). If $H = \{x^2 - x : x \in X\}$, the H-compatible assignments are all the functions $\alpha\colon X \to \{0, 1\}$. We refer to this H as *Boolean axioms.* If we are over $\mathbb{F}_2$ then, equivalently, H could be taken as $\emptyset$.

In the case of twin variables $X = \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$ and the Boolean axioms $H = \{x_i^2 - x_i, x_i + \bar{x}_i - 1 : i \in [n]\}$, the H-compatible assignments are all the functions $\alpha\colon X \to \{0, 1\}$ satisfying the additional property that for each $i \in [n]$, $\alpha(x_i) + \alpha(\bar{x}_i) = 1$. We refer to this case as *Boolean axioms with twin variables.* Similarly as before, if we are over $\mathbb{F}_2$, then, equivalently, H could be taken as $\{x_i + \bar{x}_i - 1 : i \in [n]\}$.

Sometimes a direct encoding with polynomials not coming from CNF formulas is more natural. For instance, this is the case of max-cut.

▶ **Example 2.3** (max-cut). Given a graph $G = (V, E)$ consider $X = \{x_v : v \in V\}$ and let $F = \{[\, x_{v_1} + x_{v_2} + 1 \,,\, 1\,] : (v_1, v_2) \in E\} \subseteq \mathbb{F}_2[X]$. Finding $\mathrm{cost}(F)$ is equivalent to finding a max-cut in G. This codification could be easily generalized to weighted max-cut.

3 Polynomial Calculus for MaxSAT

We first define Polynomial Calculus for MaxSAT in the special case of polynomials with coefficients in $\mathbb{F}_2$ (the general case is in Section 6). Recall that $\mathbb{F}_2$ is the finite field with 2 elements 0 and 1 (this field is unique up to isomorphism), in particular for each element of $a \in \mathbb{F}_2$, $a^2 = a$ and $2 \cdot a = a + a = 0$.

The initial instance consists of a multi-sets of weighted polynomials, i.e. pairs $[\,f\,,\,w\,]$ with $f \in \mathbb{F}_2[X]$ and $w \in \mathbb{N}$, and a set of hard polynomials H. We define $\mathbb{Z}$-*weighted Polynomial Calculus* ($\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$), an inference system that handles weighted polynomials with weights in $\mathbb{Z}$, and $\mathbb{N}$-*weighted Polynomial Calculus* ($\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$), an inference system that handles weighted polynomials with weights in $\mathbb{N}$. Both systems use the same set of inference rules. The formal definition of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}/\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ is Definition 3.3, but we discuss first the inference rules of the system. They are two types: structural rules (the FOLD, UNFOLD and the H-SIMPLIFICATION), and proper inference rules (SUM and SPLIT).

To have a sound proof system in the context of partial weighted MaxSAT, we use the inference rules as *substitution* rules, that is, when applied, these rules *replace* the premises with the conclusions. In this context, a substitution rule is *sound* if, for every assignment α, the cost of the set of premises on α equals the cost of the conclusions on α.

Fold-unfold. Let F, G be two multi-sets of weighted polynomials, we say that F and G are *fold-unfold equivalent* ($F \approx G$) if there is a sequence of multi-sets of weighted polynomials starting with F and ending with G where from one multi-set to the next, one of the following substitution rules is applied

$$\frac{[\,f\,,\,u\,] \quad [\,f\,,\,w\,]}{[\,f\,,\,u+w\,]}\ (\textsc{fold}) \qquad\qquad \frac{[\,f\,,\,u+w\,]}{[\,f\,,\,u\,] \quad [\,f\,,\,w\,]}\ (\textsc{unfold})$$

$$\frac{[\,f\,,\,0\,]}{}\ (0\text{-}\textsc{fold}) \qquad\qquad \frac{}{[\,f\,,\,0\,]}\ (0\text{-}\textsc{unfold})$$

where $f \in \mathbb{F}_2[X]$, and $w, u \in \mathbb{Z}$.

▶ **Example 3.1.** For instance, $\{[\,f\,,\,0\,]\} \approx \emptyset$ and $\{[\,f\,,\,2\,]\} \approx \{[\,f\,,\,1\,],\ [\,f\,,\,1\,]\}$.

It is immediate to see that the fold-unfold equivalence rules are sound. Notice that this fold-unfold equivalence is similar to the fold-unfold equivalence used in [6] in the context of weighted clauses and weighted Resolution.

H-equivalence. In $\mathbb{F}_2[X]$, the polynomials $x^2 - x$ and 0 are two distinct polynomials, but since they always evaluate to 0, we want to identify them. Moreover, given a set of hard constraints H, we are only interested in H-compatible assignments, hence we want to identify two polynomials f, g such that for every H-compatible assignment α, $f(\alpha) = g(\alpha)$. This can be seen as having a EQUIVALENCE rule of the form

$$\frac{[\,f\,,\,w\,]}{[\,g\,,\,w\,]}\ (H\text{-}\textsc{equivalence})$$

where $f, g \in \mathbb{F}_2[X]$ and $w \in \mathbb{Z}$ are such that for every H-compatible assignment $\alpha\colon X \to \mathbb{F}_2$, $f(\alpha) = g(\alpha)$. If f and g are H-EQUIVALENT we write $f \equiv_H g$ (when H is clear from the context we simply write $f \equiv g$). In particular, for every H, $f^2 \equiv_H f$.

Notice that, by definition, if $f \equiv_H g$ then the cost is preserved on every H-compatible α, hence the rule is sound on H-compatible αs. To efficiently check whether $f \equiv_H g$ might be problematic, depending on H.

▶ Remark 3.2. The H-equivalence could be checked efficiently for $H = \emptyset$, just by looking at the multilinearization of the polynomials. The multilinearization of a polynomial f, $\mathrm{mul}(f)$ is the unique multilinear polynomial H-equivalent to f.

In the case of polynomials coming from the direct translation of CNF formulas, we use the H-equivalence for twin variables and H being the Boolean axioms for twin variables. In this case, the H-equivalence can also be checked efficiently [19, section 4.3 and Theorem 4.4].

Sum and split. Apart from the previous structural rules, in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ and $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ we have the following inference rules:

$$\frac{[\,f\,,\,w\,]}{[\,fg\,,\,w\,]\quad[\,f(g+1)\,,\,w\,]}\ (\textsc{split})\qquad\frac{[\,f\,,\,w\,]\quad[\,g\,,\,w\,]}{[\,f+g\,,\,w\,]\quad[\,fg\,,\,2w\,]}\ (\textsc{sum})\tag{3}$$

for all $f, g \in \mathbb{F}_2[X]$ and $w \in \mathbb{Z}$.

Notice that the previous rules are *sound*. For the SPLIT rule, if $f(\alpha) = 0$ then both the conclusions are 0, but if $f(\alpha) = 1$, then exactly one of the conclusions is 1. For the SUM rule, the argument by cases is analogous. The case that justifies the weight of $2w$ for the polynomial fg in the conclusion is when $f(\alpha) = 1$ and $g(\alpha) = 1$. In this case $f(\alpha) + g(\alpha) = 0$ and $f(\alpha)g(\alpha) = 1$, hence the weight of the conclusion fg should equal the sum of the weights of both premises, which is two.

Formally the definition of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}/\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ is the following.

▶ **Definition 3.3** ($\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$, $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$). *Given a multi-set of weighted polynomials F and a set of hard constraints H, a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ derivation of a weighted polynomial $[\,f\,,\,w\,]$ from F and H is a sequence of multi-sets $L_0, \dots, L_\ell$ s.t.*

1. *$L_0 = F$,*
2. *$[\,f\,,\,w\,] \in L_\ell$ and all the other weighted polynomials $[\,f'\,,\,w'\,] \in L_\ell$ have $w' \in \mathbb{N}$, and*
3. *for each $i > 0$ either $L_i \approx L_{i-1}$ or L_i is the result of an application of the SPLIT/SUM/H-EQUIVALENCE rule as a substitution rule on L_{i-1}.*

The system $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ is the restriction of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ where all weights are natural numbers. The size of a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}/\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ derivation $L_0, \dots, L_\ell$ is the total number of occurrences of symbols in $L_0, \dots, L_\ell$.

To clarify the definition, we give an example of derivation in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$.

▶ **Example 3.4** (Example 2.3 cont.). In Fig. 2 we show a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$-derivation of $[\,1\,,\,2\,]$ from the set of polynomials we saw in Example 2.3 in the case of G being the clique on 4 vertices. That is the weighted polynomials $[\,x+y+1\,,\,1\,]$, $[\,x+z+1\,,\,1\,]$, $[\,x+t+1\,,\,1\,]$, $[\,y+z+1\,,\,1\,]$, $[\,y+t+1\,,\,1\,]$, $[\,z+t+1\,,\,1\,]$. In this derivation, the polynomials that are just copied from one multiset to the next are substituted with a $\bullet$. From one multiset to the next we applied multiple rules in parallel. The horizontal lines are just a visual help to visualize the multisets. Notice that we have H-equivalences (for $H = \emptyset$) applied implicitly. For instance, some SUM only have one consequence since the other is equivalent to 0. This example shows that to obtain $[\,1\,,\,2\,]$ it is important to use both consequences of a SUM.

We prove now the *soundness* of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$.

▶ **Theorem 3.5** (soundness). *Given $F = \{[\,f_1\,,\,w_1\,], \dots, [\,f_m\,,\,w_m\,]\}$ where $f_i \in \mathbb{F}_2[X]$ and a set of polynomials $H \subseteq \mathbb{F}_2[X]$, if there is a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ derivation of $[\,1\,,\,w\,]$ from F (and H as hard constraints), then $\mathrm{cost}_H(F) \geq w$.*

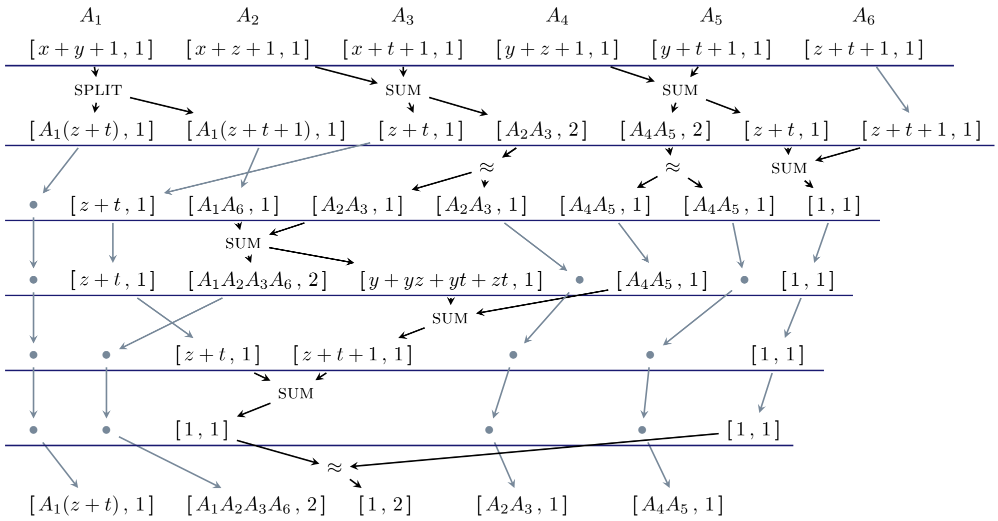


Figure 2 A $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ derivation of $[1, 2]$ from the axioms of max-cut on a clique of 4 vertices: $[x_1+x_2+1, 1]$, $[x_1+x_3+1, 1]$, $[x_1+x_4+1, 1]$, $[x_2+x_3+1, 1]$, $[x_2+x_4+1, 1]$, $[x_3+x_4+1, 1]$.

Proof. Let $L_0, L_1, L_2, \ldots, L_s$ be a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ derivation of $(1; w)$, i.e. L_s contains $[1, w]$, $L_0 = F$ and each L_{i+1} is obtained from L_i applying the SPLIT, the SUM substitution rules, the fold-unfold equivalence or the H-SIMPLIFICATION. We have to show that $\text{cost}_H(F) \geq w$. We have that $\text{cost}_H(L_s) \geq w$ since $[1, w] \in L_s$ and all the other weighted polynomials in L_s have non-negative weights. Hence, to prove the statement is enough to show that for each i, $\text{cost}_H(L_{i+1}) = \text{cost}_H(L_i)$. We prove something slightly stronger, that for each H-consistent $\alpha \colon X \to \mathbb{F}_2$, $\text{cost}(\alpha, L_{i+1}) = \text{cost}(\alpha, L_i)$. This follows immediately from the comments we already made on the soundness of the various rules. ◀

We conclude this section with an observation on the SPLIT and SUM rules in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$. Using weights in $\mathbb{Z}$, one of them is always redundant, unlike the case of weights in $\mathbb{N}$ where both are necessary. To simulate the SPLIT rule using the SUM rule using weights in $\mathbb{Z}$ we can do the following:

$$\begin{array}{c l}
[f, w] & \\
\hline
[f, w] \quad [fg, w] \quad [fg, -w] \quad [f(g+1), w] \quad [f(g+1), -w] & \approx \\
\hline
[f, w] \quad [fg, w] \quad [f(g+1), w] \quad [fg+f(g+1), -w] \quad [f^2g(g+1), -2w] & \text{SUM} \\
\hline
[fg, w] \quad [f(g+1), w] & \approx \,\&\, \equiv
\end{array}$$

In a similar way, we can also simulate the SUM using the SPLIT rule using weights in $\mathbb{Z}$. (The proof of this will appear in the final version of this paper.)

4 Completeness

We show the completeness of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$, that is the converse of Theorem 3.5. For simplicity, we focus on the Boolean axioms as hard constraints, that is $H = \emptyset$ or, in the case of twin variables $\{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$, $H = \{x_i + \bar{x}_i - 1 : i \in [n]\}$.

▶ **Theorem 4.1** (completeness for Boolean axioms). *Given F a set of weighted polynomials over $\mathbb{F}_2[X]$, there is a* $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ *derivation of* $[\,1\,,\,\mathrm{cost}_H(F)\,]$ *from F and the set of Boolean axioms as hard constraints H.*

Our proof generalizes the *saturation* process from [8] and gives an algorithm to find $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$-derivations of $[\,1\,,\,\mathrm{cost}_H(F)\,]$. We five an example of the construction in Section 5. Clearly the completeness for $H=\emptyset$ implies the completeness for $H=\{x_i+\bar{x}_i-1 : i\in[n]\}$. For instance, just removing the twin variables, that is substituting each variable $\bar{x}_i$ with $1-x_i$. We show a saturation process that adapts to this context without removing the twin variables.

Our construction shows indeed something stronger, that $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ is still complete even if we restrict the SPLIT rule of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ to be of the form

$$\frac{[\,f\,,\,w\,]}{[\,f f_{x=0} f_{x=1}\,,\,w\,]\quad[\,f(f_{x=0}f_{x=1}+1)\,,\,w\,]}\,,$$

where x is some variable and $f_{x=0}$ is the polynomial resulting from the restriction of f mapping x to 0, and analogously for $f_{x=1}$. In the case of twin variables, for $f_{x=0}$ we also map $\bar{x}$ to 1, to be consistent with the Boolean axioms, and analogously for $f_{x=1}$.

Recall that for polynomials $f,g\in\mathbb{F}_2[X]$, let $f\equiv g$ if for every H-compatible assignment α, $f(\alpha)=g(\alpha)$. It is immediate to see that $f f_{x=0}f_{x=1}\equiv f_{x=0}f_{x=1}$ since for every value $a\in\mathbb{F}_2$, $a^2=a$. Therefore, if $f\not\equiv f_{x=0}f_{x=1}$ and $f_{x=0}f_{x=1}\not\equiv 0$, the special case of the SPLIT rule above allows to infer from f some new polynomials and one of them ($f_{x=0}f_{x=1}$) without the variable x.

▶ **Definition 4.2.** *We say that a polynomial f* depends *on a variable x if for every polynomial g not containing x (and also $\bar{x}$ in the case of twin variables), $f\not\equiv g$.*

Notice that, the following are equivalent:

- f depends on x,
- For $H=\emptyset$, the multilinearization of f is a polynomial xf_1+f_0 with f_1,f_0 not containing x and $f_1\not\equiv 0$. For the twin variables and $H=\{x_i+\bar{x}_i-1 : i\in[n]\}$, f is equivalent to a multilinear polynomial of the form $xf_1+\bar{x}f_1'+f_0$, with f_1,f_1',f_0 not containing $x,\bar{x}$, and $f_1\not\equiv f_1'$.
- $f\not\equiv f_{x=0}f_{x=1}$.

(The proof of this will appear in the final version of this paper.) The reason we give these equivalences is that the third condition makes easier to generalize the whole construction to arbitrary finite fields.

The main concept used to show the completeness of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ is the notion of set of polynomials *saturated* w.r.t. a variable.

▶ **Definition 4.3** (x-saturated set). *Let $x\in X$ and S be a set of weighted polynomials. The set S is x*-saturated *if every H-compatible assignment $\alpha\colon X\to\mathbb{F}_2$ can be modified in x to a H-compatible assignment satisfying all weighted polynomials in S that depend on x.*

Notice that, if S is x-saturated then the subset of polynomials in S depending on x is satisfiable, but the converse is not true. For instance, $\{[\,x+y\,,\,1\,],\ [\,x+z\,,\,1\,]\}$ is clearly satisfiable but it is not saturated w.r.t. to x since we cannot extend the assignment $y=0,z=1$ to satisfy both polynomials.

Next, we give a procedure to x-saturate a set of weighted polynomials S. Recall that we focus on H being the Boolean axioms. Informally, the procedure to obtain the x-saturation consists of applying the SPLIT rule to a polynomial or the SUM of two polynomials, as long as the application of these rules generates polynomials that don't contain the variable x, and are not equivalent to 0. The procedure is applied as long as it is possible, and we will see that it finishes in a finite number of steps and when it terminates the generated set must be x-saturated.

▶ **Lemma 4.4.** *In the context of H the Boolean axioms, for every set of weighted polynomials S and every variable x, there is a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ derivation of a set of polynomials S' which is x-saturated.*

Proof. For a polynomial f, recall that $f_{x=0}$ is the evaluation of f in $x = 0$ and, in the case of twin variables, the restriction also sets $\bar{x} = 1$ (resp. for $f_{x=1}$).

Suppose we have a set of weighted polynomials S and a variable x. We construct a sequence of weighted polynomials $S_0, S_1, \dots$ to find the saturation. We start with $S_0 = S$, then we want S_{i+1} to be derivable from S_i using the rules of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$, and moreover, in S_{i+1} we added some new polynomial non-dependent on x. The way to obtain S_{i+1} from S_i is the following. For each $i \geq 0$, if there is an $[\,f\,,\,w\,] \in S_i$ depending on x and s.t. $f_{x=0}f_{x=1} \not\equiv 0$, non-deterministically choose one of such $[\,f\,,\,w\,]$ and let

$$S_{i+1} = (S_i \setminus \{[\,f\,,\,w\,]\}) \cup \{[\,f_{x=0}f_{x=1}\,,\,w\,],\ [\,f_{x=0}f_{x=1} + f\,,\,w\,]\}\ .$$

The derivation of S_{i+1} from S_i, by substituting $[\,f\,,\,w\,]$ with the weighted polynomials $[\,f_{x=0}f_{x=1}\,,\,w\,]$ and $[\,f_{x=0}f_{x=1} + f\,,\,w\,]$, is justified by:

$$\begin{array}{cl}
[\,f\,,\,w\,] & \\ \hline
[\,ff_{x=0}f_{x=1}\,,\,w\,] \quad [\,f(f_{x=0}f_{x=1}+1)\,,\,w\,] & \textsc{split} \\ \hline
[\,f_{x=0}f_{x=1}\,,\,w\,] \quad [\,f_{x=0}f_{x=1}+f\,,\,w\,] & \equiv
\end{array}$$

where the last $\equiv$ holds since $ff_{x=0}f_{x=1} \equiv f_{x=0}f_{x=1}$. Notice that, with this substitution, we have obtained the weighted polynomial $[\,f_{x=0}f_{x=1}\,,\,w\,]$ where the variable x doesn't appear (and hence clearly not depending on x) and it is not equivalent to 0 since the condition to obtain S_{i+1} is that $f_{x=0}f_{x=1} \not\equiv 0$. We used the assumption that f depends on x to ensure the polynomial $f_{x=0}f_{x=1}$ in the conclusions is new and it is not equivalent to the polynomial f in the premises.

If there are $[\,f\,,\,w\,], [\,g\,,\,w'\,] \in S_i$ depending on x, with $f \not\equiv g$,

$$(f_{x=0} + g_{x=0})(f_{x=1} + g_{x=1}) \not\equiv 0\ ,$$

and $w' \geq w > 0$, non-deterministically choose two of them. First substitute $[\,g\,,\,w'\,]$ by $[\,g\,,\,w\,]$ and $[\,g\,,\,w' - w\,]$, and then let

$$\begin{aligned} S_{i+1} = {} & (S_i \setminus \{[\,f\,,\,w\,],\ [\,g\,,\,w\,]\}) \cup \{[\,(f+g)_{x=0}(f+g)_{x=1}\,,\,w\,],\ [\,fg\,,\,2w\,], \\ & [\,(f+g)_{x=0}(f+g)_{x=1} + f + g\,,\,w\,]\}\ . \end{aligned}$$

We can obtain S_{i+1} from S_i using first the SUM rule to infer $[\,f+g\,,\,w\,]$ and $[\,fg\,,\,2w\,]$ and then use the SPLIT rule on $[\,f+g\,,\,w\,]$ as we did in the previous case on a single polynomial.

$$\begin{array}{cl}
[\,f\,,\,w\,] \quad [\,g\,,\,w\,] & \\ \hline
[\,f+g\,,\,w\,] \quad [\,fg\,,\,2w\,] & \textsc{sum} \\ \hline
[\,(f+g)_{x=0}(f+g)_{x=1}\,,\,w\,] \quad [\,(f+g)_{x=0}(f+g)_{x=1} + f + g\,,\,w\,] \quad [\,fg\,,\,2w\,] & \textsc{split}\ \&\ \equiv
\end{array}$$

Doing this substitution we obtain a polynomial where the variable x doesn't appear and it is not equivalent to 0 since the condition to obtain S_{i+1} is that $(f_{x=0} + g_{x=0})(f_{x=1} + g_{x=1}) \not\equiv 0$.

If we cannot transform S_i to S_{i+1} in neither of the two ways we stop the process.

This sequence of transformations must be finite and the last element S_ℓ will be x-saturated. The process must be finite since otherwise the new sequence given by

$$\sigma(i) = \sum_{\substack{\alpha\colon X\to\mathbb{F}_2\\ H\text{-compatible}}} \mathrm{cost}(\alpha, S_i^+)$$

for S_i^+ the part of S_i depending on x, would give a sequence of strictly decreasing natural numbers. Indeed, all the $\sigma(i)$s are natural numbers because we are using the rules of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$, so, no negative weight could appear in any S_i and $\mathrm{cost}(\alpha, S_i^+) \geq 0$. To show that $\sigma(i+1) < \sigma(i)$ it is sufficient to notice that, by the soundness of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$, for every H-compatible $\alpha\colon X \to \mathbb{F}_2$,

$$\mathrm{cost}(\alpha, S_{i+1}) = \mathrm{cost}(\alpha, S_i) \ ,$$

which implies that

$$\mathrm{cost}(\alpha, S_{i+1}^+) + \mathrm{cost}(\alpha, \{[\,h\,,\,w\,]\}) = \mathrm{cost}(\alpha, S_i^+) \ ,$$

for some polynomial $h \not\equiv 0$, not depending the variable x and with $w > 0$. Hence

$$\sigma(i+1) + \sum_{\substack{\alpha\colon X\to\mathbb{F}_2\\ H\text{-compatible}}} \mathrm{cost}(\alpha, \{[\,h\,,\,w\,]\}) = \sigma(i) \ ,$$

and

$$\sum_{\substack{\alpha\colon X\to\mathbb{F}_2\\ H\text{-compatible}}} \mathrm{cost}(\alpha, \{[h, w]\}) > 0$$

because $h \not\equiv 0$ and $w > 0$. Therefore, $\sigma(i+1) < \sigma(i)$. And the sequence must be finite.

Now, S_ℓ, the last set of the sequence, must be x-saturated. Suppose, towards a contradiction, that both α_0, i.e. α modified mapping $x \mapsto 0$, and α_1, i.e. α modified mapping $x \mapsto 1$, falsify some polynomials in S_ℓ depending on x. (In the case of twin variables α_0 also sets $\bar{x} \mapsto 1$ and α_1 also sets $\bar{x} \mapsto 0$.) Let such polynomials resp. be f, g. That is we have $f_{x=0}(\alpha) \neq 0$, and $g_{x=1}(\alpha) \neq 0$. Since S_ℓ is the last element of the previous process we must have that $f_{x=0}f_{x=1} \equiv 0$ and $g_{x=0}g_{x=1} \equiv 0$. Hence it must be that $f_{x=1}(\alpha) = 0$, and $g_{x=0}(\alpha) = 0$. In particular, f and g are two non-equivalent polynomials. Then, again by the assumption on S_ℓ being the last element of the process, we must also have that $(f+g)_{x=0}(f+g)_{x=1} \equiv 0$, but this is not possible since

$$(f+g)(\alpha_0) = (f+g)_{x=0}(\alpha) = f_{x=0}(\alpha) + g_{x=0}(\alpha) \neq 0$$

and similarly $f_{x=1}(\alpha) + g_{x=1}(\alpha) \neq 0$. ◀

Notice that in the proof of the previous lemma, there are many alternative ways to introduce new polynomials not depending on x in each step of the sequence $S_1, S_2, \ldots$. The one we chose has the property that we only use a special form of the SPLIT rule:

$$\frac{[\,f\,,\,w\,]}{[\,f_{x=0}f_{x=1}\,,\,w\,] \quad [\,f_{x=0}f_{x=1} + f\,,\,w\,]} \ .$$

Moreover, we keep the number and the degree of the polynomials we introduce at each step lower than other alternative choices. For instance, given multilinear polynomials $f = xf_1 + f_0$ and $g = xg_1 + g_0$ depending on x, we could have done two SPLIT to obtain $g_1 f$ and $f_1 g$ and then SUM them to obtain $f_0g_1 + f_1g_0$ (and hence introducing also $[\, f_1g_1fg\, ,\, 2\,]$). This would have resulted in the introduction of a larger number and higher degree polynomials compared to the construction we gave.

We now show how to obtain the completeness, under the assumption that the saturation can be computed in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$. Essentially iterating the saturation on all the variables one by one.

▶ **Lemma 4.5.** *Let $H \subset \mathbb{F}_2[X]$. If for every set of weighted polynomials S and every variable x, there is a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ derivation of a set of polynomials S' which is x-saturated, then for every set of weighted polynomials F over $\mathbb{F}_2[X]$ there exists a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$-derivation of $[\,1\,,\,\mathrm{cost}_H(F)\,]$ from F and the hard constraints H.*

Proof. Let $X = \{x_1, \dots, x_n\}$. First saturate F w.r.t. x_1. By hypothesis, from F in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ we can derive a x_1-saturated set S_1. Let $S_1 = S_1^+ \cup S_1^-$, where S_1^+ is the part of S_1 depending on x_1 and S_1^- the part of S_1 not depending on x_1.

Saturate S_1^- w.r.t. x_2. Again, by assumption, from S_1^- we can derive in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ a x_2-saturated set S_2. This gives a decomposition $S_2 = S_2^+ \cup S_2^-$, where S_2^- are the weighted polynomials in S_2 not depending on x_2 (and x_1). Continuing saturating by all the variables of X one by one we arrive at a set S_n, where the weighted polynomials in S_n^- are just constants, i.e. $S_n^- \approx \{[\,1\,,\,w\,]\}$ for some $w \in \mathbb{N}$.

To show that $w = \mathrm{cost}_H(F)$ it is enough to show that $\bigcup_{j\in[n]} S_j^+$ is satisfiable by a H-compatible assignment. Let $\alpha \colon X \to \mathbb{F}_2$ be an arbitrary H-compatible assignment. Since S_n is x_n-saturated there is a way to modify α in x_n to get a H-compatible assignment satisfying all S_n^+. Let this assignment be α_n. Suppose we obtained a H-compatible assignment α_i satisfying $\bigcup_{j\geq i} S_j^+$, since S_{i-1} is x_{i-1}-saturated, there is a way to modify α_i in x_{i-1} to satisfy all S_{i-1}^+. Let this assignment be α_{i-1}. Since the polynomials in $\bigcup_{j\geq i} S_j^+$ only contained the variables $x_i \dots, x_n$, the assignment α_{i-1} continues to satisfy $\bigcup_{j\geq i} S_j^+$. We continue this way until we get an assignment α_1 satisfying all $\bigcup_{j\in[n]} S_j^+$. Thus proving that it must have been that $w = \mathrm{cost}_H(F)$. ◀

Notice that the previous lemma does not require H to be the Boolean axioms and it is completely general.

Now, it is immediate to prove the completeness of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ (Theorem 4.1). Indeed, by Lemma 4.4, for every set of weighted polynomials S and every variable x, there is a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ derivation of a set of polynomials S' which is x-saturated. Then, by Lemma 4.5, there is a $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ derivation of $[\,1\,,\,\mathrm{cost}_H(F)\,]$ from F and as set of hard constraints H the Boolean axioms. This concludes the proof of Theorem 4.1.

5 Tseitin formulas

We exemplify the saturation process on *Tseitin formulas*, although also Example 3.4 was found using the saturation process. An implementation of the saturation algorithm in python is freely available at `https://github.com/jordilevy/pyPolyCal.git`.

First, we recall what are Tseitin formulas. That is, in this section, consider fixed a graph $G = (V, E)$ with $|V| = 2n + 1$ and Boolean variables $x_{v,w}$ for each $\{v, w\} \in E$. For $v \in V$, let $N(v) = \{w \in V : \{v, w\} \in E\}$. The *Tseitin formula* on G is a CNF formula expressing that in each vertex $v \in V$ the parity of the variables x_e for the edges incident to v is 1, that is Tseitin(G) is the CNF

$$\bigcup_{v\in V} \Big\{ \bigoplus_{w\in N(v)} x_{v,w} = 1 \pmod 2 \Big\}\ , \tag{4}$$

where $\bigoplus_{w\in N(v)} x_{v,w} = 1 \pmod 2$ is encoded as a set of clauses. For instance, if $N(v) = \{w_1, w_2, w_3\}$, then $\bigoplus_{w\in N(v)} x_{v,w} + 1 \pmod 2$ is

$$\begin{aligned} &\{x_{v,w_1}, x_{v,w_2}, x_{v,w_3}\},\ \{\neg x_{v,w_1}, \neg x_{v,w_2}, x_{v,w_3}\},\\ \{x_{v,w_1}, \neg x_{v,w_2}, \neg x_{v,w_3}\},\ &\{\neg x_{v,w_1}, x_{v,w_2}, \neg x_{v,w_3}\}\ . \end{aligned} \tag{5}$$

Since V has an odd size, Tseitin(G) is unsatisfiable.

Consider first the natural encoding of eq. (4) as polynomials. That is consider the set of variables $X = \{x_{v,w} : \{v, w\} \in E\}$ and L_v be the polynomial $\sum_{w\in N(v)} x_{v,w} + 1$ in $\mathbb{F}_2[X]$.

It is well known that $\mathsf{PC}_{\mathbb{F}_2}$ is able to refute $\{L_v : v \in V\}$ in linear size. In $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ we prove more.

▶ **Proposition 5.1.** *There is a linear size derivation in* $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ *of* $[\,1\,,\,c\,]$ *from* $\{[\,L_v\,,\,1\,] : v \in V\}$*, where* c *is the number of connected components of odd size in* G*. In particular, if* G *is connected,* $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ *proves that* $\{[\,L_v\,,\,1\,] : v \in V\}$ *is minimally unsatisfiable, i.e. it is possible to satisfy all polynomials in it except one.*

Proof. We show how to infer $[\,1\,,\,1\,]$ from $\{[\,L_v\,,\,1\,] : v \in V\}$ via the saturation process, when G is connected. For the saturation process, the order in which we saturate the variables is not important.

At each intermediate saturation step ℓ there is a set of weighted polynomials $\mathcal{S}_\ell$ that we have to saturate. The set $\mathcal{S}_\ell$ has the form $\{[\,L_{S_1}\,,\,1\,],\ \dots, [\,L_{S_m}\,,\,1\,]\}$, where $S_1, \dots, S_m$ form a partition of V and $L_{S_i} = \sum_{v\in S_i} L_v$. Moreover, we already saturated w.r.t. all the variables $x_{v,w}$ with v, w in the same S_i.

At the beginning of the saturation process, we have the partition of V consisting of all the singletons: $\{\{v\} : v \in V\}$.

Suppose then we are at an intermediate step ℓ of the saturation. We have a set $\mathcal{S}_\ell = \{[\,L_{S_1}\,,\,1\,],\ \dots, [\,L_{S_m}\,,\,1\,]\}$ and we want to saturate w.r.t. $x_{v,w}$. By the inductive assumption, $\{v, w\}$ is not an internal edge of any of the sets S_is. Hence there are exactly two distinct sets S_i and S_j with $v \in S_i$ and $w \in S_j$. That is, to saturate $\mathcal{S}_\ell$ w.r.t. $x_{v,w}$ is enough to saturate $\mathcal{S}' = \{[\,L_{S_i}\,,\,1\,],\ [\,L_{S_j}\,,\,1\,]\}$. We follow the procedure from Lemma 4.4.

▶ **Fact 1.** *For every* <u>*linear*</u> *polynomial* L *depending on* x*,* $L_{x=0}L_{x=1} = L_{x=0}(L_{x=0} + 1) \equiv 0$.

By Fact 1, the only possibility is to SUM L_{S_i} and L_{S_j}. That is from $\mathcal{S}'$ we obtain

$$\mathcal{S}'' = \{[\,L_{S_i} + L_{S_j}\,,\,1\,],\ [\,L_{S_i}L_{S_j}\,,\,2\,]\}\ .$$

Now, $L_{S_i} + L_{S_j} \equiv L_{S_i\cup S_j}$ does not contain variables $x_{v',w'}$ with $v', w' \in S_i \cup S_j$. In particular it does not contain $x_{v,w}$. To continue the saturation process, the only possibility would be to do a SPLIT on $L_{S_i}L_{S_j}$, but this produces the polynomial $L_{S_i,x=0}L_{S_j,x=0}L_{S_i,x=1}L_{S_j,x=1} \equiv 0$ by Fact 1. Therefore $\mathcal{S}''$ is saturated w.r.t. $x_{v,w}$. And so is the multi-set

$$\{[\,L_{S_k}\,,\,1\,] : k \neq i, j\} \cup \{[\,L_{S_i} + L_{S_j}\,,\,1\,],\ [\,L_{S_i}L_{S_j}\,,\,2\,]\}\ .$$

The part of this set not depending on $x_{v,w}$ is

$$\mathcal{S}_{\ell+1} = \{[\,L_{S_k}\,,\,1\,] : k \neq i, j\} \cup \{[\,L_{S_i} + L_{S_j}\,,\,1\,]\}\ .$$

Notice that $[\,L_{S_i}L_{S_j}\,,\,2\,]$ is not in $\mathcal{S}_{\ell+1}$ since it depends on $x_{v,w}$. The set $\mathcal{S}_{\ell+1}$ is the one we want to saturate at the next step for some other variable. During the saturation process we obtain coarser and coarser partitions of V and, at the end of the whole process, we obtain $\{[\,L_V\,,\,1\,]\}$. To conclude we just need to observe that $L_V \equiv |V| \equiv 1$. ◀

To show that $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ and weighted Resolution are incomparable, we need to consider Tseitin(G) encoded as a set of polynomials using the twin variables encoding from Section 2.2. Assume all the initial polynomials of this encoding to have weight 1. From this system of polynomials is still easy to derive $[\,1\,,\,c\,]$ in $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ where c is the number of connected components of odd size in G. Such derivations can be found using the saturation process, provided we use the natural heuristic of preferentially taking the SUM of two weighted polynomials $[\,f\,,\,w\,]$ and $[\,g\,,\,w\,]$ when $fg \equiv 0$. The intuitive reason behind this heuristic is that in such a SUM the number of polynomials in conclusion decreases and we do not introduce a polynomial of higher degree. Under this heuristic it is immediate to see that the saturation process will essentially reconstruct the polynomials $\{[\,L_v\,,\,1\,] : v \in V(G)\}$. Indeed, take for instance the twin variables encoding of the set of clauses in eq. (5), that is

$$\begin{aligned} S = \{&[\,\bar{x}_{v,w_1}\bar{x}_{v,w_2}\bar{x}_{v,w_3}\,,\,1\,],\ [\,x_{v,w_1}x_{v,w_2}\bar{x}_{v,w_3}\,,\,1\,],\\ &[\,\bar{x}_{v,w_1}x_{v,w_2}x_{v,w_3}\,,\,1\,],\ [\,x_{v,w_1}\bar{x}_{v,w_2}x_{v,w_3}\,,\,1\,]\}\ . \end{aligned}$$

We have that the product of any two of the polynomials is divisible by $x_{v,w_i}\bar{x}_{v,w_i} \equiv 0$ for some i. Therefore applying the SUM rule on S, eventually we will obtain

$$[\,\bar{x}_{v,w_1}\bar{x}_{v,w_2}\bar{x}_{v,w_3} + x_{v,w_1}x_{v,w_2}\bar{x}_{v,w_3} + \bar{x}_{v,w_1}x_{v,w_2}x_{v,w_3} + x_{v,w_1}\bar{x}_{v,w_2}x_{v,w_3}\,,\,1\,] \equiv [\,L_v\,,\,1\,]\ .$$

6 Polynomial Calculus for MaxSAT (general case)

In this section, we adapt the definition of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ and $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ from $\mathbb{F}_2$ to an arbitrary finite field $\mathbb{F}_q$. Polynomial Calculus modulo distinct primes has been studied, for instance in [11]. The Tseitin principle can be extended from counting mod 2 to counting mod p, and this principle is easy in Polynomial Calculus on polynomials with coefficients in $\mathbb{F}_p$.

That is we focus on polynomials with coefficients in $\mathbb{F}_q$, where $q = p^k$ for some fixed prime p and $k \in \mathbb{N}$. Recall that $\mathbb{F}_q$ is the finite field with q elements (this field is unique up to isomorphism), and for each element of $a \in \mathbb{F}$, $a^q = a$ and $p \cdot a = \underbrace{a + \cdots + a}_{p} = 0$.

Fold-unfold and H-equivalence. We consider multi-sets of weighted polynomials, i.e. pairs $[\,f\,,\,w\,]$ with $f \in \mathbb{F}_q[X]$ and $w \in \mathbb{Z}$, under the FOLD-UNFOLD equivalence and the H-EQUIVALENCE. This is the same as what we saw in Section 3 with the only difference that the polynomials instead of being in $\mathbb{F}_2[X]$ now belong to $\mathbb{F}_q[X]$.

Recall that, for f, g H-equivalent we write $f \equiv_H g$. In particular, for every H, $f^q \equiv_H f$. Hence, in the application of the rules, by the H-equivalence we can always assume the variables in all the polynomials to appear with degree at most $q-1$. Moreover, if $H \supseteq \{x^2 - x : x \in X\}$, then we can assume the polynomials to be multilinear. For $H = \{x^2 - x : x \in X\}$, to check efficiently if $f \equiv_H g$ we can then compute the multilinearization of f and g and compare them. In the case of the twin variables $\{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$, $H = \{x_i^2 - x_i, x_i + \bar{x}_i - 1 : i \in [n]\}$, we also have that we can check efficiently if $f \equiv_H g$ using the construction in [19, section 4.3 and Theorem 4.4].

Sum and split. The general forms of the SPLIT and SUM rules are:

$$\frac{[\,f\,,\,w\,]}{[\,fg\,,\,w\,]\quad[\,f(g^{q-1}-1)\,,\,w\,]}\qquad(\text{SPLIT})$$

$$\frac{[\,f\,,\,w\,]\quad[\,g\,,\,w\,]}{[\,f+g\,,\,w\,]\quad[\,fg\,,\,w\,]\quad[\,f\,((f+g)^{q-1}-1)\ ,\,w\,]}\qquad(\text{SUM})$$

for all $f, g \in \mathbb{F}_q[X]$ and $w \in \mathbb{Z}$. The SPLIT rule is sound since for every assignment $\alpha\colon X \to \mathbb{F}_q$ if $f(\alpha) = 0$ the cost of the premise is 0 and so is the cost of the conclusion. If $f(\alpha) \neq 0$, then either $g(\alpha) = 0$ or $g(\alpha) \neq 0$, but in this latter case then $g^{q-1}(\alpha) = 1$. The soundness of the SUM rule is analogous.

Using the rules above, we generalize immediately the definition of $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ and $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ (Definition 3.3) from weighted polynomials with coefficients in $\mathbb{F}_2$ to weighted polynomials with coefficients in $\mathbb{F}_q$. We call the resulting systems $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{N}}$ and $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{Z}}$.

Similar to the case of $\mathbb{F}_2$, we have that the SUM rule is redundant in $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{Z}}$. (The proof of this fact will appear in the final version of this paper.)

▶ **Theorem 6.1** (soundness). *Given $F = \{[\,f_1\,,\,w_1\,], \ldots, [\,f_m\,,\,w_m\,]\}$ where $f_i \in \mathbb{F}_q[X]$ and a set of polynomials $H \subseteq \mathbb{F}_q[X]$, if there is a $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{Z}}$ derivation of $[\,1\,,\,w\,]$ from F (and H as hard constraints), then $\mathrm{cost}_H(F) \geq w$.*

Proof Sketch. This is a simple generalization of the proof of Theorem 3.5. ◀

We show the completeness in the case of Boolean axioms. That is $H = \{x^2 - x : x \in X\}$ or, for the twin variables $\{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ $H = \{x_i^2 - x_i,\ x_i + \bar{x}_i - 1 : i \in [n]\}$.

▶ **Theorem 6.2** (completeness for Boolean variables). *Given F a multiset of weighted polynomials in $\mathbb{F}_q[X]$, there is a $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{N}}$ derivation of $[\,1\,,\,\mathrm{cost}_H(F)\,]$ from F, and the set of Boolean axioms as hard constraints.*

The argument is a minor adaptation of the argument we saw in Section 4. The definition of when a polynomial f depends on a variable x (Definition 4.2), the definition of (x, H)-saturated set (Definition 4.3), and Lemma 4.5 do not really depend on $\mathbb{F}_2$ and can be trivially extended to $\mathbb{F}_q$. Therefore to prove Theorem 6.2 it is enough to show how to adapt Lemma 4.4, the lemma showing how to construct the saturation.

▶ **Lemma 6.3.** *For every set of weighted polynomials S and every variable x, there is a $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{N}}$ derivation of a set of polynomials S' which is (x, H)-saturated, for H the Boolean axioms.*

Proof Sketch. The proof is analogous to the argument for Lemma 4.4. The crucial property used to prove Lemma 4.4 was that $f f_{x=0} f_{x=1} \equiv f_{x=0} f_{x=1}$, which is true for polynomials with coefficients in $\mathbb{F}_2$. For $\mathbb{F}_q$ it is analogous: the crucial property is that $f^{q-1} f_{x=0} f_{x=1} \equiv f_{x=0} f_{x=1}$. This polynomial can be derived from f using the SPLIT rule of $\mathsf{wPC}_{\mathbb{F}_q,\mathbb{N}}$ as follows

$$\begin{array}{cl} [\,f\,,\,w\,] & \\ \hline [\,f f^{q-2} f_{x=0} f_{x=1}\,,\,w\,] \quad [\,f((f^{q-2} f_{x=0} f_{x=1})^{q-1} - 1)\,,\,w\,] & \textsc{split} \\ \hline [\,f_{x=0} f_{x=1}\,,\,w\,] \quad [\,f(f_{x=0} f_{x=1})^{q-1} - f\,,\,w\,]\,, & \equiv \end{array}$$

Notice that this generalizes the case of $\mathbb{F}_2$. Similar to that special case, we have that this SPLIT is non-trivial if both $f_{x=0} f_{x=1} \not\equiv 0$ and $f^{q-2} f_{x=0} f_{x=1} \not\equiv f$, which similarly to the case of $\mathbb{F}_2$ is equivalent to saying that f depends on x. (The proof of this fact will appear in the final version of this paper.) The construction of the sequences of multi-sets is then the natural adaptation of the construction we saw for $\mathbb{F}_2$ to $\mathbb{F}_q$. The reason the sequence is finite and the obtained multi-set is x-saturated is the same as in Lemma 4.4. ◀

7 Conclusions

We showed a way to generalize Polynomial Calculus to the context of MaxSAT, for polynomials with coefficients in a finite field. This involves extending the rules of Polynomial Calculus to have additional conclusions and applying them replacing premises with conclusions, to make them sound for MaxSAT. We showed its completeness via a saturation process. The resulting proof system may be used for SAT or for MaxSAT. The system $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{N}}$ is stronger than MaxSAT Resolution, and $\mathsf{wPC}_{\mathbb{F}_2,\mathbb{Z}}$ is stronger than $\mathbb{Z}$-weighted Resolution (aka Sherali-Adams). As an example, we show how the process is able to prove efficiently Tseitin formulas.

References

1 Carlos Ansótegui, Maria Luisa Bonet, Jordi Levy, and Felip Manyà. The logic behind weighted CSP. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 32–37, 2007. URL: `http://ijcai.org/Proceedings/07/Papers/003.pdf`.

2 Carlos Ansótegui and Jordi Levy. Reducing SAT to Max2SAT. In *Proc. of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pages 1367–1373, 2021.

3 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. In *Lecture Notes in Computer Science*, pages 114–127. Springer Berlin Heidelberg, 2009.

4 Albert Atserias and Massimo Lauria. Circular (yet sound) proofs. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2019.

5 Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. Propositional proof systems based on maximum satisfiability. *Artificial Intelligence*, 300:103552, 2021.

6 Maria Luisa Bonet and Jordi Levy. Equivalence between systems stronger than resolution. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020*, pages 166–181, Cham, 2020. Springer International Publishing.

7 Maria Luisa Bonet, Jordi Levy, and Felip Manyà. A complete calculus for Max-SAT. In *Proc. of the 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 240–251, 2006.

8 Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for max-SAT. *Artif. Intell.*, 171(8-9):606–618, 2007.

9 Michael Brickenstein and Alexander Dreyer. Polybori: A framework for Groebner-basis computations with boolean polynomials. *Journal of Symbolic Computation*, 44(9):1326–1345, 2009. Effective Methods in Algebraic Geometry. `doi:10.1016/j.jsc.2008.02.017`.

10 B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19–29, August 1976. `doi:10.1145/1088216.1088219`.

11 Sam Buss, Dima Grigoriev, Russell Impagliazzo, and Toniann Pitassi. Linear gaps between degrees for the polynomial calculus modulo distinct primes. *Journal of Computer and System Sciences*, 62(2):267–289, 2001.

12 Matthew Clegg, Jeff Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 174–183. ACM, 1996.

13 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.

14 Jesús A De Loera, J. Lee, S. Margulies, and S. Onn. Expressing combinatorial problems by systems of polynomial equations and Hilbert's Nullstellensatz. *Comb. Probab. Comput.*, 18(4):551–582, July 2009. `doi:10.1017/S0963548309009894`.

15 Jesús A De Loera, Jon Lee, Peter N Malkin, and Susan Margulies. Computing infeasibility certificates for combinatorial problems through Hilbert's Nullstellensatz. *Journal of Symbolic Computation*, 46(11):1260–1283, 2011.

16 Jesús A De Loera, Susan Margulies, Michael Pernpeintner, Eric Riedl, David Rolnick, Gwen Spencer, Despina Stasi, and Jon Swenson. Graph-coloring ideals: Nullstellensatz certificates, Gröbner bases for chordal graphs, and hardness of Gröbner bases. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*, pages 133–140. ACM, 2015.

17 Susanna F. de Rezende, Massimo Lauria, Jakob Nordström, and Dmitry Sokolov. The Power of Negative Reasoning. In Valentine Kabanets, editor, *36th Computational Complexity Conference (CCC 2021)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:24, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

18 Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/780506.780516.

19 Yuval Filmus, Edward A. Hirsch, Artur Riazanov, Alexander Smal, and Marc Vinyals. Proving unsatisfiability with hitting formulas. *Electron. Colloquium Comput. Complex.*, TR23-016, 2023. arXiv:TR23-016.

20 Dima Grigoriev. Linear lower bound on degrees of positivstellensatz calculus proofs for the parity. *Theoretical Computer Science*, 259(1-2):613–622, May 2001.

21 Alexey Ignatiev, António Morgado, and João Marques-Silva. On tackling the limits of resolution in SAT solving. In *Proc. of the 20th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'17)*, pages 164–183, 2017.

22 Daniela Kaufmann, Paul Beame, Armin Biere, and Jakob Nordström. Adding dual variables to algebraic reasoning for gate-level multiplier verification. In *Proceedings of the 25th Design, Automation and Test in Europe Conference (DATE'22)*, 2022.

23 Daniela Kaufmann and Armin Biere. Nullstellensatz-proofs for multiplier verification. In *Computer Algebra in Scientific Computing - 22nd International Workshop, CASC 2020, Linz, Austria, September 14-18, 2020, Proceedings*, pages 368–389, 2020. doi:10.1007/978-3-030-60026-6_21.

24 Daniela Kaufmann, Armin Biere, and Manuel Kauers. Verifying large multipliers by combining SAT and computer algebra. In *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, pages 28–36, 2019. doi:10.23919/FMCAD.2019.8894250.

25 Daniela Kaufmann, Armin Biere, and Manuel Kauers. From DRUP to PAC and back. In *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*, pages 654–657, 2020. doi:10.23919/DATE48585.2020.9116276.

26 Javier Larrosa and Emma Rollon. Towards a better understanding of (partial weighted) MaxSAT proof systems. In *Proc. of the 23nd Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'20)*, pages 218–232, 2020.

27 Javier Larrosa and Emma Rollón. Augmenting the power of (partial) MaxSAT resolution with extension. In *Proc. of the 34th Nat. Conf. on Artificial Intelligence (AAAI'20)*, 2020.

28 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011.

29 A.A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, December 1998.

30 Emma Rollon and Javier Larrosa. Proof complexity for the maximum satisfiability problem and its use in SAT refutations. *J. Log. Comput.*, 32(7):1401–1435, 2022.

31 Dmitry Sokolov. (Semi)Algebraic proofs over $\{\pm 1\}$ variables. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing.* ACM, June 2020.

32 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, January 1987.

Certified Knowledge Compilation with Application to Verified Model Counting

Randal E. Bryant
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Wojciech Nawrocki
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA

Jeremy Avigad
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA

Marijn J. H. Heule
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Computing many useful properties of Boolean formulas, such as their weighted or unweighted model count, is intractable on general representations. It can become tractable when formulas are expressed in a special form, such as the decision-decomposable, negation normal form (dec-DNNF). *Knowledge compilation* is the process of converting a formula into such a form. Unfortunately existing knowledge compilers provide no guarantee that their output correctly represents the original formula, and therefore they cannot validate a model count, or any other computed value.

We present *Partitioned-Operation Graphs* (POGs), a form that can encode all of the representations used by existing knowledge compilers. We have designed CPOG, a framework that can express proofs of equivalence between a POG and a Boolean formula in conjunctive normal form (CNF).

We have developed a program that generates POG representations from dec-DNNF graphs produced by the state-of-the-art knowledge compiler D4, as well as checkable CPOG proofs certifying that the output POGs are equivalent to the input CNF formulas. Our toolchain for generating and verifying POGs scales to all but the largest graphs produced by D4 for formulas from a recent model counting competition. Additionally, we have developed a formally verified CPOG checker and model counter for POGs in the Lean 4 proof assistant. In doing so, we proved the soundness of our proof framework. These programs comprise the first formally verified toolchain for weighted and unweighted model counting.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases Propositional model counting, Proof checking

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.6

Supplementary Material *DataPaper (Paper Supplement)*:
`https://doi.org/10.5281/zenodo.7966174`

Funding *Randal E. Bryant*: Supported by NSF grant CCF-2108521.
Wojciech Nawrocki: Hoskinson Center for Formal Mathematics
Jeremy Avigad: Hoskinson Center for Formal Mathematics
Marijn J. H. Heule: Supported by NSF grant CCF-2108521.

1 Introduction

Given a Boolean formula ϕ, modern Boolean satisfiability (SAT) solvers can find an assignment satisfying ϕ or generate a proof that no such assignment exists. They have applications across a variety of domains including computational mathematics, combinatorial optimization, and the formal verification of hardware, software, and security protocols. Some applications, however, require going beyond Boolean satisfiability. For example, the *model counting problem*

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 6; pp. 6:1–6:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

requires computing the number of satisfying assignments of a formula, including in cases where there are far too many to enumerate individually. Model counting has applications in artificial intelligence, computer security, and statistical sampling. There are also many useful extensions of standard model counting, including *weighted model counting*, where a weight is defined for each possible assignment, and the goal becomes to compute the sum of the weights of the satisfying assignments.

Model counting is a challenging problem – more challenging than the already NP-hard Boolean satisfiability. Several tractable variants of Boolean satisfiability, including 2-SAT, become intractable when the goal is to count models and not just determine satisfiability [28]. Nonetheless, a number of model counters that scale to very large formulas have been developed, as witnessed by the progress in recent model counting competitions.

One approach to model counting, known as *knowledge compilation*, transforms the formula into a structured form for which model counting is straightforward. For example, the *deterministic, decomposable, negation normal form* (det-DNNF) introduced by Darwiche [7, 8], as well as the more restricted *decision-DNNF* (dec-DNNF) [17, 24], represent a Boolean formula as a directed acyclic graph, with terminal nodes labeled by Boolean variables and their complements, and with each nonterminal node labeled by a Boolean And or Or operation. Restrictions are placed on the structure of the graph (described in Section 5) such that a count of the models can be computed by a single bottom-up traversal. Kimmig et al. present a very general *algebraic model counting* [19] framework describing properties of Boolean functions that can be efficiently computed from a det-DNNF representation. These include standard and weighted model counting, and much more.

One shortcoming of existing knowledge compilers is that they have no generally accepted way to validate that the compiled representation is logically equivalent to the original formula. By contrast, all modern SAT solvers can generate checkable proofs when they encounter unsatisfiable formulas. The guarantee provided by a checkable certificate of correctness enables users of SAT solvers to fully trust their results. Experience has also shown that being able to generate proofs allow SAT solver developers to quickly detect and diagnose bugs in their programs. This, in turn, has led to more reliable SAT solvers.

This paper introduces *Partitioned-Operation Graphs* (POGs), a form that can encode all of the representations produced by current knowledge compilers. The CPOG (for "certified" POG) file format then captures both the structure of a POG and a checkable proof of its logical equivalence to a Boolean formula in conjunctive normal form (CNF). A CPOG proof consists of a sequence of clause addition and deletion steps, based on an extended resolution proof system [27]. We establish a set of conditions that, when satisified by a CPOG file, guarantees that it encodes a well-formed POG and provides a valid equivalence proof.

Figure 1 illustrates our certifying knowledge compilation and model counting toolchain. Starting with input formula ϕ_I, the D4 knowledge compiler [20] generates a dec-DNNF representation, and the *proof generator* uses this to generate a CPOG file. The *proof checker* verifies the equivalence of the CNF and CPOG representations. The *ring evaluator* computes a standard or weighted model count from the POG representation. As the dashed box in Figure 1 indicates, this toolchain moves the root of trust away from the complex and highly optimized knowledge compiler to a relatively simple checker and evaluator. Importantly, the proof generator need not be trusted – its errors will be caught by the proof checker.

To ensure soundness of the abstract CPOG proof system, as well as correctness of its concrete implementation, we formally verified the proof system as well as versions of the proof checker and ring evaluator in the Lean 4 proof assistant [11]. Running these two programs on a CPOG file gives strong assurance that the proof and the model count are correct. Our

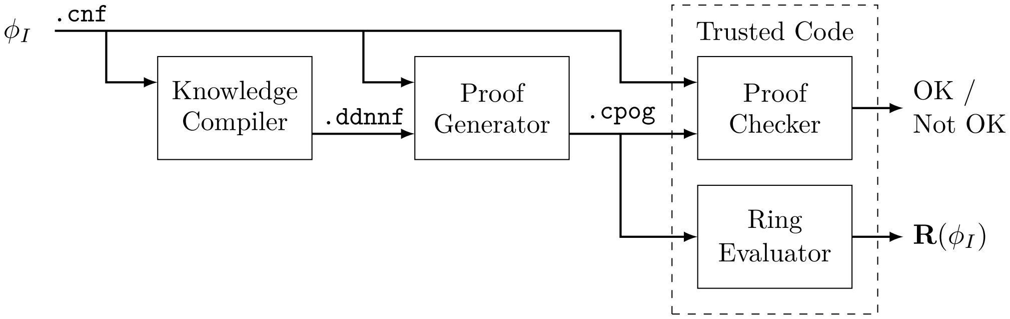


Figure 1 Certifying toolchain. The output of a standard knowledge compiler is converted into a combined graph/proof (CPOG) which can be independently checked and evaluated.

experience with developing a formally verified proof checker has shown that, even within the well-understood framework of extended resolution, it can be challenging to formulate a full set of requirements that guarantee soundness. In fact, our efforts to formally verify our proof framework exposed subtle conditions that we had to impose on our partitioned sum rule.

We evaluate our toolchain using benchmark formulas from the 2022 standard and weighted model competitions. Our tools handle all but the largest dec-DNNF graphs generated by D4. We also measure the benefits of several optimizations as well as the relative performance of the verified checker with one designed for high performance and capacity.

We also show that our tools can provide end-to-end verification of formulas that have been transformed by an equivalence-preserving preprocessor. That is, verification is based on the original formula, and so proof checking certifies correct operation of the preprocessor, the knowledge compiler, and the proof generator.

We have developed an online supplement to this paper [3] that includes a worked example, more details on the algorithms, and extensive experimental results.

2 Related Work

Generating proofs of unsatisfiability in SAT solvers has a long tradition [31] and has become widely accepted due to the formulation of clausal proof systems for which proofs can readily be generated and efficiently checked [15, 30]. A number of formally verified checkers have been developed within different verification frameworks [6, 14, 21, 26]. The associated proofs add clauses while preserving satisfiability until the empty clause is derived. Our work builds on the well-established technology and tools associated with clausal proof systems, but we require features not found in proofs of unsatisfiability. Our proofs construct a new propositional formula, and we must verify that it is equivalent to the input formula. This requires verifying additional proof steps, including clause deletion steps, and subtle invariants, as described in Sections 7 and 10.

Capelli, Lagniez, and Marquis developed a knowledge compiler that generates a certificate in a proof system that is itself based on dec-DNNF [4, 5]. Their CD4 program, a modified version of D4, generates annotations to the compiled representation, providing information about how the compiled version relates to the input clauses. It also generates a file of clausal proof steps in the DRAT format [30]. Completing the certification involves running two different checkers on the annotated dec-DNNF graph and the DRAT file. Although the authors make informal arguments regarding the soundness of their frameworks, these do not provide strong levels of assurance. Indeed, we have identified a weakness in their methodology due to an invalid assumption about the guarantees provided by DRAT-TRIM, the program it uses to check the DRAT file. This weakness is *exploitable*: their framework can be "spoofed" into accepting an incorrect compilation.

In more detail, CD4 emits a sequence of clauses R that includes the conflict clauses that arose during a top-down processing of the input clauses. Given input formula ϕ_I, their first task is to check whether $\phi_I \Rightarrow R$, i.e., that any assignment that satisfies ϕ_I also satisfies each of the clauses in R. They then base other parts of their proof on that property and use a separate program to perform a series of additional checks. They use DRAT-TRIM to prove the implication, checking that each clause in R satisifies the *reverse asymmetric tautology* (RAT) property with respect to the preceding clauses [15, 18]. Adding a RAT clause C to a set of clauses maintains satisfiability, but it does not necessarily preserve models. As an example, consider the following formulas:

$$\begin{array}{lll} \phi_1: & (x_1 \lor x_3) & \\ \phi_2: & (x_1 \lor x_3) \quad \land & (x_2 \lor \overline{x}_3) \end{array}$$

Clearly, these two formulas are not equivalent – ϕ_1 has six models, while ϕ_2 has four. In particular, ϕ_1 allows arbitrary assignments to variable x_2. Critically, however, the second clause of ϕ_2 is RAT with respect to ϕ_1 – any satisfying assignment to ϕ_1 can be transformed into one that also satisfies ϕ_2 by setting x_2 to 1, while keeping the values for other variables fixed.

This weakness would allow a buggy (or malicious) version of CD4 to spoof the checking framework. Given formula ϕ_1 as input, it could produce a compiled result, including annotations, based on ϕ_2 and also include the second clause of ϕ_2 in R. The check with DRAT-TRIM would pass, as would the other tests performed by their checker. We have confirmed this possibility with their compiler and checker.[1]

This weakness can be corrected by restricting DRAT-TRIM to only add clauses that obey the stronger *reverse unit propagation* (RUP) property [13, 29]. We have added a command-line argument to DRAT-TRIM that enforces this restriction.[2] This weakness, however, illustrates the general challenge of developing a new proof framework. As we can attest, without engaging in an effort to formally verify the framework, there are likely to be conditions that make the framework unsound.

Fichte, Hecher, and Roland [12] devised the MICE proof framework for model counting programs. Their proof rules are based on the algorithms commonly used by model counters. They developed a program that can generate proof traces from dec-DNNF graphs and a program to check adherence to their proof rules. This framework is not directly comparable to ours, since it only certifies the unweighted model count, but it has similar goals. Again, they provide only informal arguments regarding the soundness of their framework.

Both of these prior certification frameworks are strongly tied to the algorithms used by the knowledge compilers and model counters. Some of the conditions to be checked are relevant only to specific implementations. Our framework is very general and is based on a small set of proof rules. It builds on the highly developed concepts of clausal proof systems. These factors were important in enabling formal verification. In Section 12 and the supplement [3], we also compare the performance of our toolchain to these other two.

[1] Downloaded May 18, 2023 as `https://github.com/crillab/d4/tree/333370cc1e843dd0749c1efe88516e72b5239174`.

[2] Available at `https://github.com/marijnheule/drat-trim/releases/tag/v05.22.2023`.

3 Logical Foundations

Let X denote a set of Boolean variables, and let α be an *assignment* of truth values to some subset of the variables, where 0 denotes false and 1 denotes true, i.e., $\alpha \colon X' \rightarrow \{0,1\}$ for some $X' \subseteq X$. We say the assignment is *total* when it assigns a value to every variable ($X' = X$), and that it is *partial* otherwise. The set of all possible total assignments over X is denoted $\mathcal{U}$.

For each variable $x \in X$, we define the *literals* x and $\overline{x}$, where $\overline{x}$ is the negation of x. An assignment α can be viewed as a set of literals, where we write $\ell \in \alpha$ when $\ell = x$ and $\alpha(x) = 1$ or when $\ell = \overline{x}$ and $\alpha(x) = 0$. We write the negation of literal ℓ as $\overline{\ell}$. That is, $\overline{\ell} = \overline{x}$ when $\ell = x$ and $\overline{\ell} = x$ when $\ell = \overline{x}$.

▶ **Definition 1** (Boolean Formulas). *The set of Boolean formulas is defined recursively. Each formula ϕ has an associated* dependency set $\mathcal{D}(\phi) \subseteq X$*, and a set of* models $\mathcal{M}(\phi)$*, consisting of total assignments that satisfy the formula:*

1. *Boolean constants* 0 *and* 1 *are Boolean formulas, with $\mathcal{D}(0) = \mathcal{D}(1) = \emptyset$, with $\mathcal{M}(0) = \emptyset$, and with $\mathcal{M}(1) = \mathcal{U}$.*
2. *Variable x is a Boolean formula, with $\mathcal{D}(x) = \{x\}$ and $\mathcal{M}(x) = \{\alpha \in \mathcal{U} | \alpha(x) = 1\}$.*
3. *For formula ϕ, its* negation*, written $\neg\phi$ is a Boolean formula, with $\mathcal{D}(\neg\phi) = \mathcal{D}(\phi)$ and $\mathcal{M}(\neg\phi) = \mathcal{U} - \mathcal{M}(\phi)$.*
4. *For formulas $\phi_1, \phi_2, \ldots, \phi_k$, their* product $\phi = \bigwedge_{1 \leq i \leq k} \phi_i$ *is a Boolean formula, with $\mathcal{D}(\phi) = \bigcup_{1 \leq i \leq k} \mathcal{D}(\phi_i)$ and $\mathcal{M}(\phi) = \bigcap_{1 \leq i \leq k} \mathcal{M}(\phi_i)$.*
5. *For formulas $\phi_1, \phi_2, \ldots, \phi_k$, their* sum $\phi = \bigvee_{1 \leq i \leq k} \phi_i$ *is a Boolean formula, with $\mathcal{D}(\phi) = \bigcup_{1 \leq i \leq k} \mathcal{D}(\phi_i)$ and $\mathcal{M}(\phi) = \bigcup_{1 \leq i \leq k} \mathcal{M}(\phi_i)$.*

We highlight some special classes of Boolean formulas. A formula is in *negation normal form* when negation is applied only to variables. A formula is in *conjunctive normal form* (CNF) when i) it is in negation normal form, and ii) sum is applied only to literals. A CNF formula can be represented as a set of *clauses*, each of which is a set of literals. Each clause represents the sum of the literals, and the formula is the product of its clauses. We use set notation to reference the clauses within a formula and the literals within a clause. A clause consisting of a single literal is referred to as a *unit* clause and the literal as a *unit* literal. This literal must be assigned value 1 by any satisfying assignment of the formula.

▶ **Definition 2** (Partitioned-Operation Formula). *A* partitioned-operation formula *satisfies the following for all product and sum operations:*

1. *The arguments to each product must have disjoint dependency sets. That is, operation $\bigwedge_{1 \leq i \leq k} \phi_i$ requires $\mathcal{D}(\phi_i) \cap \mathcal{D}(\phi_j) = \emptyset$ for $1 \leq i < j \leq k$.*
2. *The arguments to each sum must have disjoint models. That is, operation $\bigvee_{1 \leq i \leq k} \phi_i$ requires $\mathcal{M}(\phi_i) \cap \mathcal{M}(\phi_j) = \emptyset$ for $1 \leq i < j \leq k$.*

We let $\wedge^{\mathsf{p}}$ and $\vee^{\mathsf{p}}$ denote the product and sum operations in a partitioned-operation formula.

4 Ring Evaluation of a Boolean Formula

We propose a general framework for summarizing properties of Boolean formulas along the lines of algebraic model counting [19].

▶ **Definition 3** (Commutative Ring). *A* commutative ring $\mathcal{R}$ *is an algebraic structure $\langle \mathcal{A}, +, \times, \boldsymbol{0}, \boldsymbol{1} \rangle$, with elements in the set $\mathcal{A}$ and with commutative and associative operations $+$ (addition) and $\times$ (multiplication), such that multiplication distributes over addition. $\boldsymbol{0}$ is the additive identity and $\boldsymbol{1}$ is the multiplicative identity. Every element $a \in \mathcal{A}$ has an* additive inverse $-a$ *such that $a + -a = \boldsymbol{0}$.*

We write $a - b$ as a shorthand for $a + -b$.

▶ **Definition 4** (Ring Evaluation Problem). *For commutative ring $\mathcal{R}$, a* ring weight function *associates a value $w(x) \in \mathcal{A}$ with every variable $x \in X$. We then define $w(\overline{x}) \doteq \boldsymbol{1} - w(x)$.*

For Boolean formula ϕ and ring weight function w, the ring evaluation problem *computes*

$$\boldsymbol{R}(\phi, w) \quad = \quad \textstyle\sum_{\alpha \in \mathcal{M}(\phi)} \; \prod_{\ell \in \alpha} w(\ell) \tag{1}$$

In this equation, sum $\sum$ is computed using addition operation $+$, and product $\prod$ is computed using multiplication operation $\times$.

Many important properties of Boolean formulas can be expressed as ring evaluation problems. The (standard) *model counting* problem for formula ϕ requires determining $|\mathcal{M}(\phi)|$. It can be cast as a ring evaluation problem by having $+$ and $\times$ be addition and multiplication over rational numbers and using weight function $w(x) = 1/2$ for every variable x. Ring evaluation of formula ϕ gives the *density* of the formula, i.e., the fraction of all possible total assignments that are models. For $n = |X|$, scaling the density by 2^n yields the number of models. This formulation avoids the need for a "smoothing" operation, in which redundant expressions are inserted into the formula [10].

The *weighted model counting* problem is also defined over rational numbers. Some formulations allow independently assigning weights $W(x)$ and $W(\overline{x})$ for each variable x and its complement, with the possibility that $W(x) + W(\overline{x}) \neq 1$. We can cast this as a ring evaluation problem by letting $r(x) = W(x) + W(\overline{x})$, performing ring evaluation with weight function $w(x) = W(x)/r(x)$ for each variable x, and computing the weighted count as $\mathbf{R}(\phi, w) \; \times \; \prod_{x \in X} r(x)$. Of course, this requires that $r(x) \neq 0$ for all $x \in X$.

The *function hashing problem* provides a test of inequivalence for Boolean formulas. That is, for $n = |X|$, let $\mathcal{R}$ be a finite field with $|\mathcal{A}| = m$ such that $m \geq 2n$. For each $x \in X$, choose a value from $\mathcal{A}$ at random for $w(x)$. Two formulas ϕ_1 and ϕ_2 will clearly have $\mathbf{R}(\phi_1, w) = \mathbf{R}(\phi_2, w)$ if they are logically equivalent, and if $\mathbf{R}(\phi_1, w) \neq \mathbf{R}(\phi_2, w)$, then they are clearly inequivalent. If they are not equivalent, then the probability that $\mathbf{R}(\phi_1, w) \neq \mathbf{R}(\phi_2, w)$ will be at least $\left(1 - \frac{1}{m}\right)^n \geq \left(1 - \frac{1}{2n}\right)^n > 1/2$. Function hashing can therefore be used as part of a randomized algorithm for equivalence testing [2]. For example, it can compare different runs on a single formula, either from different compilers or from a single compiler with different configuration parameters.

5 Partitioned-Operation Graphs (POGs)

Performing ring evaluation on an arbitrary Boolean formula could be intractable, but it is straightforward for a formula with partitioned operations:

▶ **Proposition 5.** *Ring evaluation with operations $\neg$, $\wedge^{\mathsf{p}}$, and $\vee^{\mathsf{p}}$ satisfies the following for any weight function w:*

$$\begin{array}{rcl}
\boldsymbol{R}(\neg\phi,\ w) & = & \boldsymbol{1} - \boldsymbol{R}(\phi, w) \\
\boldsymbol{R}\left(\bigwedge^{\mathsf{p}}{}_{1 \leq i \leq k} \phi_i,\ w\right) & = & \prod_{1 \leq i \leq k} \boldsymbol{R}(\phi_i, w) \\
\boldsymbol{R}\left(\bigvee^{\mathsf{p}}{}_{1 \leq i \leq k} \phi_i,\ w\right) & = & \sum_{1 \leq i \leq k} \boldsymbol{R}(\phi_i, w)
\end{array}$$

As is described in Section 10, we have proved these three equations using Lean 4.

A *partitioned-operation graph* (POG) is a directed, acyclic graph with nodes N and edges $E \subseteq N \times N$. We denote nodes with boldface symbols, such as $\mathbf{u}$ and $\mathbf{v}$. When $(\mathbf{u}, \mathbf{v}) \in E$, node $\mathbf{v}$ is said to be a *child* of node $\mathbf{u}$. The in- and out-degrees of node $\mathbf{u}$ are defined as indegree($\mathbf{u}$) = $|E \cap (N \times \{\mathbf{u}\})|$, and outdegree($\mathbf{u}$) = $|E \cap (\{\mathbf{u}\} \times N)|$. Node $\mathbf{u}$ is said to be *terminal* if outdegree($\mathbf{u}$) = 0. A terminal node is labeled by a Boolean constant or variable. Node $\mathbf{u}$ is said to be *nonterminal* if outdegree($\mathbf{u}$) > 0. A nonterminal node is labeled by Boolean operation $\wedge^{\mathsf{p}}$ or $\vee^{\mathsf{p}}$. A node can be labeled with operation $\wedge^{\mathsf{p}}$ or $\vee^{\mathsf{p}}$ only if it satisfies the partitioning restriction for that operation. Every POG has a designated *root node* $\mathbf{r}$. Each edge has a *polarity*, indicating whether (negative polarity) or not (positive polarity) the corresponding argument should be negated.

A POG represents a partitioned-operation formula with a sharing of common subformulas. Every node in the graph can be viewed as a partitioned-operation formula, and so we write $\phi_{\mathbf{u}}$ as the formula denoted by node $\mathbf{u}$. Each such formula has a set of models, and we write $\mathcal{M}(\mathbf{u})$ as a shorthand for $\mathcal{M}(\phi_{\mathbf{u}})$.

We can now define and compare two related representations:

- A det-DNNF graph can be viewed a POG with negation applied only to variables.
- A dec-DNNF graph is a det-DNNF graph with the further restriction that any sum node $\mathbf{u}$ has exactly two children $\mathbf{u}_1$ and $\mathbf{u}_0$, and for these there is a *decision variable* x such that any total assignment $\alpha \in \mathcal{M}(\mathbf{u}_b)$ has $\alpha(x) = b$, for $b \in \{0, 1\}$.

The generalizations encompassed by POGs have also been referred to as *deterministic decomposable circuits* (d-Ds) [23]. Our current proof generator only works for knowledge compilers generating dec-DNNF representations, but these generalizations allow for future extensions, while maintaining the ability to efficiently perform ring evaluation.

We define the *size* of POG P, written $|P|$, to be the the number of nonterminal nodes plus the number of edges from these nodes to their children. Ring evaluation of P can be performed with at most $|P|$ ring operations by traversing the graph from the terminal nodes up to the root, computing a value $\mathbf{R}(\mathbf{u}, w)$ for each node $\mathbf{u}$. The final result is then $\mathbf{R}(\mathbf{r}, w)$.

6 Clausal Proof Framework

We write (possibly subscripted) θ for formulas encoded as clauses, possibly with extension variables. We write (possibly subscripted) ϕ for formulas that use no extension variables.

A proof in our framework consists of a sequence of clause addition and deletion steps, with each step preserving the set of solutions to the original formula. The status of the proof at any step is represented as a set of *active* clauses θ, i.e., those that have been added but not yet deleted. Our framework is based on *extended* resolution [27], where proof steps can introduce new *extension variables* encoding Boolean formulas over input and prior extension variables. Let Z denote the set of extension variables occuring in formula θ. Starting with θ equal to input formula ϕ_I, the proof must maintain the invariant that $\phi_I \Leftrightarrow \exists Z\, \theta$.

Clauses can be added in two different ways. One is when they serve as the *defining clauses* for an extension variable. This form occurs only when defining $\wedge^{\mathsf{p}}$ and $\vee^{\mathsf{p}}$ operations, as is described in Section 7. Clauses can also be added or deleted based on *implication redundancy*. That is, when clause C satisfies $\theta \Rightarrow C$ for formula θ, then it can either be added to θ to create the formula $\theta \cup \{C\}$ or it can be deleted from $\theta \cup \{C\}$ to create θ.

We use *reverse unit propagation* (RUP) to certify implication redundancy when adding or deleting clauses [13, 29]. RUP is the core rule supported by standard proof checkers [15, 30] for propositional logic. It provides a simple and efficient way to check a sequence of applications of the resolution proof rule [25]. Let $C = \{\ell_1, \ell_2, \ldots, \ell_p\}$ be a clause to be proved redundant with respect to formula θ. Let $D_1, D_2, \ldots, D_k$ be a sequence of supporting *antecedent* clauses, such that each D_i is in θ. A RUP step proves that $\bigwedge_{1 \leq i \leq k} D_i \Rightarrow C$ by showing that the combination of the antecedents plus the negation of C leads to a contradiction. The negation of C is the formula $\overline{\ell}_1 \wedge \overline{\ell}_2 \wedge \cdots \wedge \overline{\ell}_p$, having a CNF representation consisting of p unit clauses of the form $\overline{\ell}_i$ for $1 \leq i \leq p$. A RUP check processes the clauses of the antecedent in sequence, inferring additional unit clauses. In processing clause D_i, if all but one of the literals in the clause is the negation of one of the accumulated unit clauses, then we can add this literal to the accumulated set. That is, all but this literal have been falsified, and so it must be set to true for the clause to be satisfied. The final step with clause D_k must cause a contradiction, i.e., all of its literals are falsified by the accumulated unit clauses.

Compared to the proofs of unsatisfiability generated by SAT solvers, ours have important differences. Most significantly, each proof step must preserve the set of solutions with respect to the input variables; our proofs must therefore justify both clause deletions and additions. By contrast, an unsatisfiability proof need only guarantee that no proof step causes a satisfiable set of clauses to become unsatisfiable, and therefore it need only justify clause additions.

7 The CPOG Representation and Proof System

A CPOG file provides both a declaration of a POG, as well as a checkable proof that a Boolean formula, given in conjunctive normal form, is logically equivalent to the POG. The proof format draws its inspiration from the LRAT [14] and QRAT [16] formats for unquantified and quantified Boolean formulas, respectively. Key properties include:

- The file contains declarations of $\wedge^{\mathsf{p}}$ and $\vee^{\mathsf{p}}$ operations to describe the POG. Declaring a node $\mathbf{u}$ implicitly adds an *extension* variable u and a set of *defining* clauses θ_u encoding the product or sum operation. This is the only means for adding extension variables to the proof.
- Boolean negation is supported implicitly by allowing the arguments of the $\vee^{\mathsf{p}}$ and $\wedge^{\mathsf{p}}$ operations to be literals and not just variables.
- The file contains explicit clause addition steps. A clause can only be added if it is logically implied by the existing clauses. A sequence of clause identifiers must be listed as a *hint* providing a RUP verification of the implication.
- The file contains explicit clause deletion steps. A clause can only be deleted if it is logically implied by the remaining clauses. A sequence of clause identifiers must be listed as a *hint* providing a RUP verification of the implication.
- The checker must track the dependency set for every input and extension variable. For each $\wedge^{\mathsf{p}}$ operation, the checker must ensure that the dependency sets for its arguments are disjoint. The associated extension variable has a dependency set equal to the union of those of its arguments.
- Declaring a $\vee^{\mathsf{p}}$ operation requires a sequence of clauses providing a RUP proof that the arguments are mutually exclusive. Only binary $\vee^{\mathsf{p}}$ operations are allowed to avoid requiring multiple proofs of disjointness

7.1 Syntax

Table 1 shows the declarations that can occur in a CPOG file. As with other clausal proof formats, a variable is represented by a positive integer v, with the first ones being input variables and successive ones being extension variables. Literal ℓ is represented by a signed integer, with $-v$ being the logical negation of variable v. Each clause is indicated by a positive integer identifier C, with the first ones being the IDs of the input clauses and successive ones being the IDs of added clauses. Clause identifiers must be totally ordered, such that any clause identifier C' given in the hint when adding clause C must have $C' < C$.

Table 1 CPOG Step Types. C: clause identifier, L: literal, V: variable.

Rule				Description
C	`a`	L^* `0`	C^+ `0`	Add RUP clause
	`d`	C	C^+ `0`	Delete RUP clause
C	`p`	$V\ L^*$ `0`		Declare $\wedge^{\mathsf{p}}$ operation
C	`s`	$V\ L\ L$	C^+ `0`	Declare $\vee^{\mathsf{p}}$ operation
	`r`	L		Declare root literal

Table 2 Defining Clauses for Product (A) and Sum (B) Operations.

(A). Product Operation $\wedge^{\mathsf{p}}$

ID	Clause				
i	v	$-\ell_1$	$-\ell_2$	$\cdots$	$-\ell_k$
$i{+}1$	$-v$	ℓ_1			
$i{+}2$	$-v$	ℓ_2			
	$\cdots$				
$i{+}k$	$-v$	ℓ_k			

(B). Sum Operation $\vee^{\mathsf{p}}$

ID	Clause		
i	$-v$	ℓ_1	ℓ_2
$i{+}1$	v	$-\ell_1$	
$i{+}2$	v	$-\ell_2$	

The first set of proof rules are similar to those in other clausal proofs. Clauses can be added via RUP addition (command `a`), with a sequence of antecedent clauses (the "hint"). Similarly for clause deletion (command `d`).

The declaration of a *product* operation, creating a node with operation $\wedge^{\mathsf{p}}$, has the form:

$$i \quad \mathtt{p} \quad v \quad \ell_1 \quad \ell_2 \quad \cdots \quad \ell_k \quad \mathtt{0}$$

Integer i is a new clause ID, v is a positive integer that does not correspond to any previous variable, and $\ell_1, \ell_2, \ldots, \ell_k$ is a sequence of k integers, indicating the arguments as literals of existing variables. As Table 2(A) shows, this declaration implicitly causes $k+1$ clauses to be added to the proof, providing a Tseitin encoding that defines extension variable v as the product of its arguments.

The dependency sets for the arguments represented by each pair of literals ℓ_i and ℓ_j must be disjoint, for $1 \leq i < j \leq k$. A product operation may have no arguments, representing Boolean constant 1. The only clause added to the proof will be the unit literal v. A reference to literal $-v$ then provides a way to represent constant 0.

The declaration of a *sum* operation, creating a node with operation $\vee^{\mathsf{p}}$, has the form:

$$i \quad \mathtt{s} \quad v \quad \ell_1 \quad \ell_2 \;\; H \quad \mathtt{0}$$

Integer i is a new clause ID, v is a positive integer that does not correspond to any previous variable, and ℓ_1 and ℓ_2 are signed integers, indicating the arguments as literals of existing variables. Hint H consists of a sequence of clause IDs, all of which must be defining clauses for other POG operations.[3] As Table 2(B) shows, this declaration implicitly causes three

[3] The restriction to defining clauses in the hint is critical to soundness. Allowing the hint to include the IDs of input clauses creates an exploitable weakness. We discovered this weakness in the course of our efforts at formal verification.

clauses to be added to the proof, providing a Tseitin encoding that defines extension variable v as the sum of its arguments. The hint must provide a RUP proof of the clause $\overline{\ell}_1 \vee \overline{\ell}_2$, showing that the two children of this node have disjoint models.

Finally, the literal denoting the root of the POG is declared with the `r` command. It can occur anywhere in the file. Except in degenerate cases, it will be the extension variable representing the root of a graph.

7.2 Semantics

The defining clauses for a product or sum operation uniquely define the value of its extension variable for any assignment of values to the argument variables. For the extension variable u associated with any POG node $\mathbf{u}$, we can therefore prove that any total assignment α to the input variables that also satisfies the POG defining clauses will assign a value to u such that $\alpha(u) = 1$ if and only if $\alpha \in \mathcal{M}(\mathbf{u})$.

The sequence of operator declarations, asserted clauses, and clause deletions represents a systematic transformation of the input formula into a POG. Validating all of these steps serves to prove that POG P is logically equivalent to the input formula. At the completion of the proof, the following FINAL CONDITIONS must hold:

1. There is exactly one remaining clause that was added via RUP addition, and this is a unit clause consisting of root literal r.
2. All of the input clauses have been deleted.

In other words, at the end of the proof it must hold that the active clauses be exactly those in $\theta_P \doteq \{\{r\}\} \cup \bigcup_{\mathbf{u} \in P} \theta_u$, the formula consisting of unit clause $\{r\}$ and the defining clauses for the nodes, providing a Tseitin encoding of P. Recognizing that any total assignment to the input variables implicitly defines the assignments to the extension variables, we can see that θ_P is the clausal encoding of $\phi_{\mathbf{r}}$. Let ϕ_I denote the input formula. The sequence of clause addition steps provides a *forward implication* proof that $\mathcal{M}(\phi_I) \subseteq \mathcal{M}(\phi_{\mathbf{r}})$. That is, any total assignment α satisfying the input formula must also satisfy the formula represented by the POG. Conversely, each proof step that deletes an input clause proves that any total assignment α that falsifies the clause must falsify $\phi_{\mathbf{r}}$. Deleting all but the final asserted clause and all input clauses provides a *reverse implication* proof that $\mathcal{M}(\phi_{\mathbf{r}}) \subseteq \mathcal{M}(\phi_I)$.

The supplement [3], shows the CPOG description for an input formula with five clauses, yielding a POG with six nonterminal nodes. It explains how the clause addition and deletion steps yield a proof of equivalence between the input formula and its POG representation.

8 Generating CPOG from dec-DNNF

A dec-DNNF graph can be directly translated into a POG. In doing this conversion, our program performs simplifications to eliminate Boolean constants. Except in degenerate cases, where the formula is unsatisfiable or a tautology, we can therefore assume that the POG does not contain any constant nodes. In addition, negation is only applied to variables, and so the only edges with negative polarity will have variables as children. We can therefore view the POG as consisting of *literal* nodes corresponding to input variables and their negations, along with *nonterminal* nodes, which can be further classified as *product* and *sum* nodes.

8.1 Forward Implication Proof

For input formula ϕ_I and its translation into a POG P with root node $\mathbf{r}$, the most challenging part of the proof is to show that $\mathcal{M}(\phi_I) \subseteq \mathcal{M}(\phi_{\mathbf{r}})$, i.e., that any total assignment α that is a model of ϕ_I and the POG definition clauses will yield $\alpha(r) = 1$, for root literal r. This part of the proof consists of a series of clause assertions leading to one adding $\{r\}$ as a unit clause. We have devised two methods of generating this proof. The *monolithic* approach makes just one call to a proof-generating SAT solver and has it determine the relationship between the two representations. The *structural* approach recursively traverses the POG, generating proof obligations at each node encountered. It may require multiple calls to a proof-generating SAT solver.

As notation, let ψ be a subset of the clauses in ϕ_I. For partial assignment ρ, the expression $\psi|_\rho$ denotes the set of clauses γ obtained from ψ by: i) eliminating any clause containing a literal ℓ such that $\rho(\ell) = 1$, ii) for the remaining clauses eliminating those literals ℓ for which $\rho(\ell) = 0$, and iii) eliminating any duplicate or tautological clauses. In doing these simplifications, we also track the *provenance* of each simplified clause C, i.e., which of the (possibly multiple) input clauses simplified to become C. More formally, for $C \in \psi|_\rho$, we let $\text{Prov}_\rho(C, \psi)$ denote those clauses $C' \in \psi$, such that $C' \subseteq C \cup \bigcup_{\ell\in\rho} \overline{\ell}$. We then extend the definition of Prov to any simplified formula γ as $\text{Prov}_\rho(\gamma, \psi) = \bigcup_{C\in\gamma} \text{Prov}_\rho(C, \psi)$.

The monolithic approach takes advantage of the clausal representations of the input formula ϕ_I and the POG formula $\phi_{\mathbf{r}}$. We can express the negation of $\phi_{\mathbf{r}}$ in clausal form as $\theta_{\overline{\mathbf{r}}} \doteq \bigcup_{\mathbf{u}\in P} \theta_u|_{\{\overline{r}\}}$. Forward implication will hold when $\phi_I \Rightarrow \phi_{\mathbf{r}}$, or equivalently when the formula $\phi_I \wedge \theta_{\overline{\mathbf{r}}}$ is unsatisfiable, where the conjunction can be expressed as the union of the two sets of clauses. The proof generator writes the clauses to a file and invokes a proof-generating SAT solver. For each clause C in the unsatisfiability proof, it adds clause $\{r\} \cup C$ to the CPOG proof, and so the empty clause in the proof becomes the unit clause $\{r\}$. Our experimental results show that this approach can be very effective and generates short proofs for smaller problems, but it does not scale well enough for general use.

We describe the structural approach to proof generation as a recursive procedure $\mathsf{validate}(\mathbf{u}, \rho, \psi)$ taking as arguments POG node $\mathbf{u}$, partial assignment ρ, and a set of clauses $\psi \subseteq \phi_I$. The procedure adds a number of clauses to the proof, culminating with the addition of the *target* clause: $u \vee \bigvee_{\ell\in\rho} \overline{\ell}$, indicating that $(\bigwedge_{\ell\in\rho} \ell) \Rightarrow u$, i.e., that any total assignment α such that $\rho \subseteq \alpha$ will assign $\alpha(u) = 1$. The top-level call has $\mathbf{u} = \mathbf{r}$, $\rho = \emptyset$, and $\psi = \phi_I$. The result will therefore be to add unit clause $\{r\}$ to the proof. Here we present a correct, but somewhat inefficient formulation of $\mathsf{validate}$. We then refine it with some optimizations.

The recursive call $\mathsf{validate}(\mathbf{u}, \rho, \psi)$ assumes that we have traversed a path from the root node down to node $\mathbf{u}$, with the literals encountered in the product nodes forming the partial assignment ρ. The set of clauses ψ can be a proper subset of the input clauses ϕ_I when a product node has caused a splitting into clauses containing disjoint variables. The subgraph with root node $\mathbf{u}$ should be a POG representation of the formula $\psi|_\rho$.

The process for generating such a proof depends on the form of node $\mathbf{u}$:

1. If $\mathbf{u}$ is a literal ℓ', then the formula $\psi|_\rho$ must consist of the single unit clause $C = \{\ell'\}$, such that any $C' \in \text{Prov}_\rho(C, \psi)$ must have $C' \subseteq \{\ell'\} \cup \bigcup_{\ell\in\rho} \overline{\ell}$. Any of these can serve as the target clause.
2. If $\mathbf{u}$ is a sum node with children $\mathbf{u}_1$ and $\mathbf{u}_0$, then, since the node originated from a dec-DNNF graph, there must be some variable x such that either $\mathbf{u}_1$ is a literal node for x or $\mathbf{u}_1$ is a product node containing a literal node for x as a child. In either case, we recursively call $\mathsf{validate}(\mathbf{u}_1, \rho \cup \{x\}, \psi)$. This will cause the addition of the target clause $u_1 \vee \overline{x} \vee \bigvee_{\ell\in\rho} \overline{\ell}$. Similarly, either $\mathbf{u}_0$ is a literal node for $\overline{x}$ or $\mathbf{u}_0$ is a product node containing a literal node for $\overline{x}$ as a child. In either case, we recursively call $\mathsf{validate}(\mathbf{u}_0, \rho \cup \{\overline{x}\}, \psi)$, causing the addition of the target clause $u_0 \vee x \vee \bigvee_{\ell\in\rho} \overline{\ell}$. These recursive results can be combined with the second and third defining clauses for $\mathbf{u}$ (see Table 2(B)) to generate the target clause for $\mathbf{u}$, requiring at most two RUP steps.
3. If $\mathbf{u}$ is a product node, then we can divide its children into a set of literal nodes λ and a set of nonterminal nodes $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$.

a. For each literal $\ell \in \lambda$, we must prove that any total assignment α, such that $\rho \subset \alpha$ has $\alpha(\ell) = 1$. In some cases, this can be done by simple Boolean constraint propagation (BCP). In other cases, we must prove that the formula $\psi|_{\rho \cup \{\bar{\ell}\}}$ is unsatisfiable. We do so by writing the formula to a file, invoking a proof-generating SAT solver, and then converting the generated unsatisfiability proof into a sequence of clause additions in the CPOG file. (The solver is constrained to only use RUP inference rules, preventing it from introducing extension variables.)

b. For a single nonterminal child ($k = 1$), we recursively call $\mathsf{validate}\,(\mathbf{u}_1, \rho \cup \lambda, \psi)$.

c. For multiple nonterminal children ($k > 1$), it must be the case that the clauses in $\gamma = \psi|_\rho$ can be partitioned into k subsets $\gamma_1, \gamma_2, \ldots, \gamma_k$ such that $\mathcal{D}(\gamma_i) \cap \mathcal{D}(\gamma_j) = \emptyset$ for $1 \leq i < j \leq k$, and we can match each node $\mathbf{u}_i$ to subset γ_i based on its literals. For each i such that $1 \leq i \leq k$, let $\psi_i = \mathrm{Prov}_\rho(\gamma_i, \psi)$, i.e., those input clauses in ψ that, when simplified, became clause partition γ_i. We recursively call $\mathsf{validate}\,(\mathbf{u}_i, \rho \cup \lambda, \psi_i)$.

We then generate the target clause for node $\mathbf{u}$ with a single RUP step, creating the hint by combining the results from the BCP and SAT calls for the literals, the recursively computed target clauses, and all but the first defining clause for node $\mathbf{u}$ (see Table 2(A)).

Observe that all of these steps involve a polynomial number of operations per recursive call, with the exception of those that call a SAT solver to validate a literal.

8.2 Reverse Implication Proof

Completing the equivalence proof of input formula ϕ_I and its POG representation with root node $\mathbf{r}$ requires showing that $\mathcal{M}(\phi_{\mathbf{r}}) \subseteq \mathcal{M}(\phi_I)$. This is done in the CPOG framework by first deleting all asserted clauses, except for the final unit clause for root literal r, and then deleting all of the input clauses.

The asserted clauses can be deleted in reverse order, using the same hints that were used in their original assertions. By reversing the order, those clauses that were used in the hint when a clause was added will still remain when it is deleted.

Each input clause deletion can be done as a single RUP step, based on an algorithm to test for clausal entailment in det-DNNF graphs [4, 10]. The proof generator constructs the hint sequence from the defining clauses of the POG nodes via a single, bottom-up pass through the graph. The RUP deletion proof for input clause C effectively proves that any total assignment α that satisfies the POG definition clauses but does not satisfy C will yield $\alpha(r) = 0$. It starts with the set of literals $\{\bar{\ell} \mid \ell \in C\}$, describing the required condition for assignment α to falsify clause C. It then adds literals via unit propagation until a conflict arises. Unit literal r gets added right away, setting up a potential conflict.

Working upward through the graph, node $\mathbf{u}$ is *marked* when the collected set of literals forces u to evaluate to 0. When marking $\mathbf{u}$, the program adds $\overline{u}$ to the RUP literals and adds the appropriate defining clause to the hint. A literal node for ℓ will be marked if $\ell \in C$, with no hint required. If product node $\mathbf{u}$ has some child $\mathbf{u}_i$ that is marked, then $\mathbf{u}$ is marked and clause $i + 1$ from among its defining clauses (see Table 2(A)) is added to the hint. Marking sum node $\mathbf{u}$ requires that its two children are marked. The first defining clause for this node (see Table 2(B)) will then be added to the hint. At the very end, the program (assuming the reverse implication holds) will attempt to mark root node $\mathbf{r}$, which would require $\alpha(r) = 0$, yielding a conflict.

It can be seen that the reverse implication proof will be polynomial in the size of the POG, because each clause deletion requires a single RUP step having a hint with length bounded by the number of POG nodes.

9 Optimizations

The performance of the structural proof generator for forward implication, both in its execution time and the size of the proof generated, can be improved by two optimizations described here. A key feature is that they do not require any changes to the proof framework – they build on the power of extended resolution to enable new logical structures. They involve declaring new product nodes to encode products of literals. These nodes are not part of the POG representation of the formula; they serve only to enable the forward implication proof. Here we summarize the two optimizations. The supplement [3] provides more details.

Literal Grouping: A single recursive step of validate can encounter product nodes having many – tens or even hundreds – of literals as children. The earlier formulation of validate considers each literal $\ell \in \lambda$ separately, calling a SAT solver for every literal that cannot be validated with BCP. Literal grouping handles all of these literals together. It defines product node $\mathbf{v}$ having the literals as children. The goal then becomes to prove that any total assignment must yield 1 for extension variable v. Calling a solver with v set to 0 yields an unsatisfiability proof that can be mapped back to a sequence of clause additions in the CPOG file validating all of the literals.

Lemmas: Our formulation of validate requires each call at a node $\mathbf{u}$ to recursively validate all of its children. This effectively expands the graph into a tree, potentially requiring an exponential number of recursive steps. Instead, for each node $\mathbf{u}$ having indegree$(\mathbf{u}) > 1$, the program can define and generate the proof of a *lemma* for $\mathbf{u}$ when it is first reached by a call to validate and then apply this lemma for this and subsequent calls. The lemma states that the POG with root node $\mathbf{u}$ satisfies forward implication for a formula $\gamma_{\mathbf{u}}$, where some of the clauses in this formula are input clauses from ϕ_I, but others are simplified versions of input clauses. The key idea is to introduce product nodes to encode (via DeMorgan's Laws) the simplified clauses and have these serve as lemma arguments.

The combination of these two optimization guarantees that i) each call to validate for a product node will cause at most one invocation of the SAT solver, and ii) each call to validate for any node $\mathbf{u}$ will cause further recursive calls only once. Our experimental results [3] show that these optimizations yield substantial benefits.

10 A Formally Verified Toolchain

We set out to formally verify the system with two goals in mind: first, to ensure that the CPOG framework is mathematically sound; and second, to implement correct-by-construction proof checking and ring evaluation (the "Trusted Code" components of Figure 1). These two goals are achieved with a single proof development in the Lean 4 programming language [11]. Verification was greatly aided by the Aesop [22] automated tactic. In this section, we briefly describe the functionality we implemented and what we proved about it. More information is provided in the supplement [3].

Proof checking. The goal of a CPOG proof is to construct a POG that is equivalent to the input CNF ϕ_I. The POG being constructed, and the set of active clauses are stored in the checker state `st`. The checker begins by parsing the input formula, initializing the active clauses to $\theta \leftarrow \phi_I$, and initializing the POG P to an empty one. It then processes every step of the CPOG proof, either modifying its state by adding/deleting clauses in θ and adding nodes to P, or throwing an exception if a step is incorrect. Afterwards, it carries out the FINAL CONDITIONS check of Section 7.2. Throughout the process, we maintain invariants that ensure that P is partitioned and that a successful final check entails the logical equivalence of ϕ_I and $\phi_{\mathbf{r}}$, where $\mathbf{r}$ is the final POG root (Theorem 6).

The specifications we use to state these invariants are built on a general theory of propositional logic, mirroring Section 3. Following the DIMACS CNF convention, we define the data types `Var` of variables being positive natural numbers, `ILit` of literals being non-zero integers, and `PropForm` of propositional formulas. `PropForm` is generic over the type of variables, so we instantiate it with our type as `PropForm Var`. Assignments of truth values are taken to be total functions `PropAssignment Var := Var → Bool`. Requiring totality is not a limitation: instead of talking about two equal, partial assignments to a subset $X' \subseteq X$ of variables, we can more conveniently talk about two total assignments that agree on X'. We write $\sigma \models \varphi$ when `σ : PropAssignment Var` satisfies `φ : PropForm Var`.

The invariants refer to the checker state `st` with fields `st.inputCnf` for ϕ_I, `st.clauseDb` for θ, `st.pog` for P, `st.pogDefsForm` for the clausal POG definitions formula $\bigwedge_{\mathbf{u}\in P} \theta_u$, and `st.allVars` for all variables (original and extension) added so far. For any $\mathbf{u} \in P$, `st.pog.toPropForm u` computes $\phi_{\mathbf{u}}$. The first two invariants state that assignments to original variables extend uniquely to extension variables defining the POG nodes. In the formalization, we split this into extension and uniqueness:

```
/-- Any assignment satisfying φ₁ extends to φ₂ while preserving values on X. -/
def extendsOver (X : Set Var) (φ₁ φ₂ : PropForm Var) :=
  ∀ (σ₁ : PropAssignment Var), σ₁ ⊨ φ₁ → ∃ σ₂, σ₁.agreeOn X σ₂ ∧ σ₂ ⊨ φ₂
/-- Assignments satisfying φ are determined on Y by their values on X. -/
def uniqueExt (X Y : Set Var) (φ : PropForm Var) :=
  ∀ (σ₁ σ₂ : PropAssignment Var), σ₁ ⊨ φ → σ₂ ⊨ φ → σ₁.agreeOn X σ₂ →
    σ₁.agreeOn Y σ₂

invariants.extends_pogDefsForm : extendsOver st.inputCnf.vars ⊤ st.pogDefsForm
invariants.uep_pogDefsForm : uniqueExt st.inputCnf.vars st.allVars st.pogDefsForm
```

Note that in the definition of `uniqueExt`, the arrows associate to the right, so the definition says that the three assumptions imply the conclusion. The next invariant guarantees that the set of solutions over the original variables is preserved:

```
def equivalentOver (X : Set Var) (φ₁ φ₂ : PropForm Var) :=
  extendsOver X φ₁ φ₂ ∧ extendsOver X φ₂ φ₁

invariants.equivInput : equivalentOver st.inputCnf.vars st.inputCnf st.clauseDb
```

Finally, for every node $\mathbf{u} \in P$ with corresponding literal u we ensure that $\phi_{\mathbf{u}}$ is partitioned (Definition 2) and relate $\phi_{\mathbf{u}}$ to its clausal encoding $\theta_u \doteq u \wedge \bigwedge_{\mathbf{v}\in P} \theta_v$:

```
def partitioned : PropForm Var → Prop
  | tr | fls | var _ => True
  | neg φ     => φ.partitioned
  | disj φ ψ => φ.partitioned ∧ ψ.partitioned ∧ ∀ τ, ¬(τ ⊨ φ ∧ τ ⊨ ψ)
  | conj φ ψ => φ.partitioned ∧ ψ.partitioned ∧ φ.vars ∩ ψ.vars = ∅

invariants.partitioned : ∀ (u : ILit), (st.pog.toPropForm u).partitioned
invariants.equivalent_lits : ∀ (u : ILit), equivalentOver st.inputCnf.vars
    (u ∧ st.pogDefsForm) (st.pog.toPropForm x)
```

These invariants are maintained by valid CPOG proofs. Together with additional invariants that ensure the correctness of cached computations, they imply the soundness theorem for P with root node $\mathbf{r}$:

▶ **Theorem 6.** *If the proof checker has assembled POG P with root node $\mathbf{r}$ starting from input formula ϕ_I, and* FINAL CONDITIONS *(as stated in Section 7.2) hold of the checker state, then ϕ_I is logically equivalent to $\phi_{\mathbf{r}}$.*

Proof. FINAL CONDITIONS imply that the active clausal formula θ is exactly $\theta_P \doteq \{\{r\}\} \cup \bigcup_{\mathbf{u} \in P} \theta_u$. The conclusion follows from this and the checker invariants. The full proof is formally verified in Lean. ◀

After certifying a CPOG proof, the checker can pass its in-memory POG representation to the ring evaluator, along with the partitioning guarantee provided by `invariants.partitioned`.

Ring evaluation. We formalized the central quantity (1) in the ring evaluation problem (Definition 4) in a commutative ring `R` as follows:

```
def weightSum {R : Type} [CommRing R]
    (weight : Var → R) (φ : PropForm Var) (s : Finset Var) : R :=
  Σ τ in models φ s, ∏ x in s, if τ x then weight x else 1 - weight x
```

The rules for efficient ring evaluation of partitioned formulas are expressed as:

```
def ringEval (weight : Var → R) : PropForm Var → R
  | tr        => 1
  | fls       => 0
  | var x     => weight x
  | neg φ     => 1 - ringEval weight φ
  | disj φ ψ => ringEval weight φ + ringEval weight ψ
  | conj φ ψ => ringEval weight φ * ringEval weight ψ
```

Proposition 5 is then formalized as follows:

```
theorem ringEval_eq_weightSum (weight : Var → R) {φ : PropForm Var} :
    partitioned φ → ringEval weight φ = weightSum weight φ (vars φ)
```

To efficiently compute the ring evaluation of a formula represented by a POG node, we implemented `Pog.ringEval` and then proved that it matches the specification above:

```
theorem ringEval_eq_ringEval (pog : Pog) (weight : Var → R) (x : Var) :
  pog.ringEval weight x = (pog.toPropForm x).ringEval weight
```

Applying this to the output of our verified CPOG proof checker, which we know to be partitioned and equivalent to the input formula ϕ_I, we obtain a proof that our toolchain computes the correct ring evaluation of ϕ_I.

Model counting. Finally, we established that ring evaluation with the appropriate weights corresponds to the standard model count. To do so, we defined a function that carries out an integer calculation of the number of models over a set of variables of cardinality `nVars`:

```
def countModels (nVars : Nat) : PropForm Var → Nat
  | tr        => 2^nVars
  | fls       => 0
  | var _     => 2^(nVars - 1)
  | neg φ     => 2^nVars - countModels nVars φ
  | disj φ ψ => countModels nVars φ + countModels nVars ψ
  | conj φ ψ => countModels nVars φ * countModels nVars ψ / 2^nVars
```

We then formally proved that for a partitioned formula whose variables are among a finite set `s`, this computation really does count the number of models over `s`:

```
theorem countModels_eq_card_models {φ : PropForm Var} {s : Finset Var} :
  vars φ ⊆ s → partitioned φ → countModels (card s) φ = card (models φ s)
```

In particular, taking `s` to be exactly the variables of φ, we have that the number of models of φ on its variables is `countModels φ (card (vars φ))`.

11 Implementations

We have implemented programs that, along with the D4 knowledge compiler, form the toolchain illustrated in Figure 1.[4] The proof generator is the same in both cases, since it need not be trusted. Our *verified* version of the proof checker and ring evaluator have been formally verified within the Lean 4 theorem prover. Our long term goal is to rely on these. Our *prototype* version is written in C. It is faster and more scalable, but we anticipate its need will diminish as the verified version is further optimized.

Our proof generator is written in C/C++ and uses CaDiCal [1] as its SAT solver. To convert proof steps back into hinted CPOG clause additions, the generator can use either its own RUP proof generator, or it can invoke drat-trim [15]. The latter yields shorter proofs and scales well to large proofs, but each invocation has a high startup cost. We therefore only use it when solving larger problems (currently ones with over 1000 clauses).

The proof generator can optionally be instructed to generate a *one-sided* proof, providing only the reverse-implication portion of the proof via input clause deletion. This can provide useful information – any assignment that is a model for the compiled representation must also be a model for the input formula – even when full validation is impractical.

We incorporated a ring evaluator into the prototype checker. It can perform both standard and weighted model counting with full precision. It performs arithmetic over a subset of the rationals we call $\mathbf{Q}_{2,5}$, consisting of numbers of the form $a \cdot 2^b \cdot 5^c$, for integers a, b, and c, and with a implemented to have arbitrary range. Allowing scaling by powers of 2 enables the density computation and rescaling required for standard model counting. Allowing scaling by powers of both 2 and 5 enables exact decimal arithmetic, handling the weights used in the weighted model counting competitions. To give a sense of scale, the counter generated a result with 260,909 decimal digits for one of the weighted benchmarks.

12 Experimental Evaluation

We summarize the results of our experiments here. The supplement [3] provides a more complete description. For our evaluation, we used the public benchmark problems from the 2022 standard and weighted model competitions.[5] We found that there were 180 unique CNF files among these, ranging in size from 250 to 2,753,207 clauses. We ran our programs on a processor with 64 GB of memory and having an attached high-speed, solid-state disk. With a runtime limit of 4000 seconds, D4 completed for 124 of the benchmark problems. Our proof generator was able to convert all of these into POGs, with their declarations ranging from 304 to 2,761,457,765 (median 848,784) defining clauses.

[4] The source code for all tools is available at `https://github.com/rebryant/cpog/releases/tag/v1.0.0`.

[5] Downloaded from `https://mccompetition.org/2022/mc_description.html`

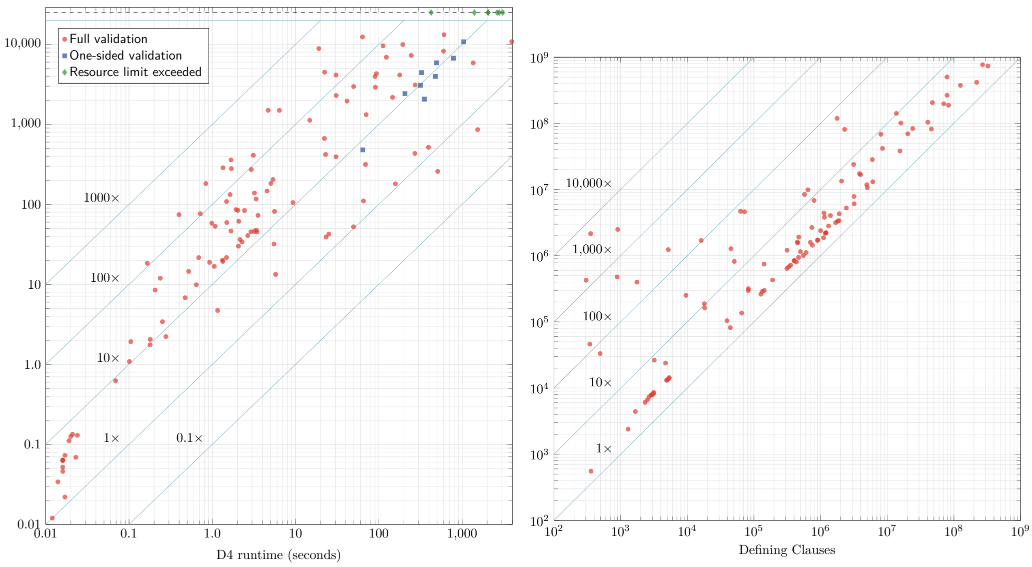


Figure 2 Runtime (left) and proof size (right) for CPOG proofs. The runtime includes proof generation, checking, and model counting relative to the runtime for D4. The proof size is measured as total clauses relative to the number of defining clauses.

We ran our proof generator with a time limit of 10,000 seconds. The results are shown in Figure 2, The left-hand plot shows the elapsed time for the combination of proof generation, checking, and counting versus the time for D4. The proof generator was able to generate full proofs for 108 of the problems and one-sided proofs for an additional 9 of them, leaving just 7 with no verification. The prototype checker successfully verified all of the generated proofs. The longest runtime for the combination of proof generation, checking, and counting for a full proof was 13,145 seconds. Overall the ratio between the combined time for generation, checking, and counting versus the time for D4 had a harmonic mean of 5.5. The right-hand plot shows the total number of clauses in the CPOG file versus the number of defining clauses for the problems having full proofs. The ratio between the total number of clauses and the number of defining clauses had a harmonic mean of 3.13. To date, we have not found any errors in the dec-DNNF graphs generated by D4.

We found that the monolithic approach for generating the forward implication proof works well for smaller POGs (up to one million defining clauses), but it becomes inefficient for larger ones. These experiments suggest a possible hybrid approach, stopping the recursion of the structural approach and shifting to monolithic mode once the subgraph size is below some threshold. By using monolithic mode, we were also able to perform end-to-end verification of all but one of the benchmarks that could be verified without preprocessing, with a total time limit (including preprocessing) of 1,000 seconds.

We also found that our two optimizations: literal grouping and lemmas can provide substantial improvements in proof size and runtime. In the extreme cases, a lack of lemmas caused one proof to grow by a factor of 52.5, while a lack of literal grouping caused another proof to grow by a factor of 39.6. Overall, it is clear that these two optimizations are worthwhile, and sometimes critical, to success for some benchmarks.

Running the verified proof checker in Lean 4 required, on average (harmonic mean), around 5.9 times longer than the prototype checker. Encouragingly, the scaling trends were identical for the two solvers, indicating that the two checkers have similar asymptotic performance. We consider a factor of 5.9 to be acceptable for the assurance provided by formal verification.

In comparing other proof frameworks, we found that the CD4 toolchain ran very fast and could handle very large benchmarks. Even with a total time limit of 1,000 seconds, including the time for knowledge compilation, the CD4 toolchain completed 106 benchmarks, while the CPOG toolchain completed just 82. We found that the runtimes for the MICE toolchain versus our CPOG toolchain showed little correlation, reflecting the fact that the two solve different problems and use different approaches. In general, our CPOG toolchain showed better scaling, in part due to its ability to control the recursion through lemmas. Neither of these prior toolchains could perform end-to-end verification when the knowledge compilation was preceded by preprocessing.

13 Conclusions

This paper demonstrates a method for certifying the equivalence of two different representations of a Boolean formula: an input formula represented in conjunctive normal form, and a compiled representation that can then be used to extract useful information about the formula, including its weighted and unweighted model counts. It builds on the extensive techniques that have been developed for clausal proof systems, including extended resolution and reverse unit propagation, as well as established tools, such as proof-generating SAT solvers and DRAT-TRIM.

We are hopeful that having checkable proofs for knowledge compilers will allow them to be used in applications where high levels of trust are required, and that it will provide a useful tool for developers of knowledge compilers. Our experiments demonstrate that our toolchain can already handle problems nearly at the limits of current knowledge compilers. Further engineering and optimization of our proof generator and checker could improve their performance and capacity substantially. We are hopeful that our tool can be adapted to handle other knowledge compiler representations, such as sentential decision diagrams (SDDs) [9].

References

1 Armin Biere. CaDiCaL at the SAT Race 2019. In *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*, pages 8–9. University of Helsinki, 2019.

2 Manuel Blum, Ashok K. Chandra, and Mark N. Wegman. Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10(2):80–82, 18 March 1980.

3 Randal E. Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Supplement to certified knowledge compilation with application to verified model counting, May 2023. doi:10.5281/zenodo.7966174.

4 Florent Capelli. Knowledge compilation languages as proof systems. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 11628 of *LNCS*, pages 90–91, 2019.

5 Florent Capelli, Jean-Marie Lagniez, and Pierre Marquis. Certifying top-down decision-DNNF compilers. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

6 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In *Conference on Automated Deduction (CADE)*, volume 10395 of *LNCS*, pages 220–236, 2017.

7 Adnan Darwiche. A compiler for deterministic, decomposable negation normal form. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2002.

8 Adnan Darwiche. New advances in compiling CNF to decomposable negation normal form. In *European Conference on Artificial Intelligence*, pages 328–332, 2004.

9 Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *International Joint Conference on Artificial Intelligence*, pages 819–826, 2011.

10 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17, 2002.

11 Leonardo de Moura and Sebastian Ulrich. The Lean 4 theorem prover and programming language. In *Conference on Automated Deduction (CADE)*, volume 12699 of *LNAI*, pages 625–635, 2021.

12 Johannes K Fichte, Markus Hecher, and Valentin Roland. Proofs for propositional model counting. In *Theory and Applications of Satisfiability Testing (SAT)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

13 Evgueni I. Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Design, Automation and Test in Europe (DATE)*, pages 886–891, 2003.

14 Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Nathan D. Wetzler. Efficient, verified checking of propositional proofs. In *Interactive Theorem Proving*, volume 10499 of *LNCS*, pages 269–284, 2017.

15 Marijn J. H. Heule, Warren A. Hunt, Jr., and Nathan D. Wetzler. Verifying refutations with extended resolution. In *Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pages 345–359, 2013.

16 Marijn J. H. Heule, Martina Seidl, and Armin Biere. A unified proof system for QBF preprocessing. In *International Joint Conference on Automated Reasoning (IJCAR)*, volume 8562 of *LNCS*, pages 91–106, 2014.

17 Jinbo Huang and Adnan Darwiche. The language of search. *Journal of Artificial Intelligence Research*, 22:191–219, 2007.

18 Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *International Joint Conference on Automated Reasoning (IJCAR)*, volume 7364 of *LNCS*, pages 355–370, 2012.

19 Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *Journal of Applied Logic*, 22:46–62, July 2017.

20 Jean-Marie Lagniez and Pierre Marquis. An improved decision-DNNF compiler. In *International Joint Conference on Artificial Intelligence*, pages 667–673, 2017.

21 Peter Lammich. Efficient verified (UN)SAT certificate checking. *J. Autom. Reason.*, 64(3):513–532, 2020. `doi:10.1007/s10817-019-09525-z`.

22 Jannis Limperg and Asta Halkjær From. Aesop: White-box best-first proof search for Lean. In *Certified Programs and Proofs (CPP)*, pages 253–266. ACM, 2023. `doi:10.1145/3573105.3575671`.

23 Mikaël Monet and Dan Olteanu. Towards deterministic decomposable circuits for safe queries. In *Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)*, 2018.

24 Umut Oztok and Adnan Darwiche. On compiling CNF into decision-DNNF. In *Constraint Programming (CP)*, volume 8656 of *LNCS*, pages 42–57, 2014.

25 John A. Robinson. A machine-oriented logic based on the resolution principle. *J.ACM*, 12(1):23–41, January 1965.

26 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation redundancy checking in CakeML. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part II*, volume 12652 of *LNCS*, pages 223–241, 2021.

27 G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer, 1983.

28 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3):410–421, 1979.

29 Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *Proc. of the 10th Int. Symposium on Artificial Intelligence and Mathematics (ISAIM 2008)*, 2008.

30 Nathan D. Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 8561 of *LNCS*, pages 422–429, 2014.

31 Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Design, Automation and Test in Europe (DATE)*, pages 880–885, 2003.

Separating Incremental and Non-Incremental Bottom-Up Compilation

Alexis de Colnet
Algorithms and Complexity Group, TU Wien, Austria

Abstract

The aim of a compiler is, given a function represented in some language, to generate an equivalent representation in a target language L. In bottom-up (BU) compilation of functions given as CNF formulas, constructing the new representation requires compiling several subformulas in L. The compiler starts by compiling the clauses in L and iteratively constructs representations for new subformulas using an "Apply" operator that performs conjunction in L, until all clauses are combined into one representation. In principle, BU compilation can generate representations for any subformulas and conjoin them in any way. But an attractive strategy from a practical point of view is to augment one main representation – which we call the core – by conjoining to it the clauses one at a time. We refer to this strategy as incremental BU compilation. We prove that, for known relevant languages L for BU compilation, there is a class of CNF formulas that admit BU compilations to L that generate only polynomial-size intermediate representations, while their incremental BU compilations all generate an exponential-size core.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Knowledge Compilation, Bottom-up Compilation, DNNF, OBDD

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.7

Funding *Alexis de Colnet*: This work has been supported by the Austrian Science Fund (FWF), ESPRIT project FWF ESP 235.

1 Introduction

Knowledge compilation (KC) is a domain of computer sciences that deals with different models for representing knowledge, or functions. Here, a *compilation* is a procedure that changes the representation of a function into something that allows for efficient reasoning. One aspect of KC is to find classes of representations, or *compilation languages*, that render interesting queries tractable [6, 15]. For Boolean functions, many such languages are subsets of the class of circuits in decomposable negation normal form (DNNF) [5].

When the function to compile into a sublanguage L of DNNF is given as a system of constraints, say a CNF formula (where constraints are clauses), different compilation paradigms are available, in particular top-down (TD) compilation and bottom-up (BU) compilation. TD compilers produce a compiled form as the trace of an algorithm that explores the space of solutions to the system, for instance the trace of a DPLL algorithm that does not stop after finding a solution but instead keep searching for more [11, 20]. A more general description is that TD compilers start from the whole system of constraints and recursively consider smaller systems. In contrast, BU compilers first compile each constraint independently, and then combine their compiled representations in pairs to construct a representation of the whole system. A key component in this paradigm is an "Apply" operator that, given two functions written in L, efficiently constructs a representation of their conjunction in L. Since binary conjunction is tractable, under some conditions, for ordered binary decision diagrams (OBDD) [2], for sentential decision diagrams (SDD) [4], and more generally for circuits in structured decomposable negation normal form (strDNNF) [19], in practice the target languages of BU compilers are restricted to sublanguages of strDNNF (including OBDD and SDD). A inconvenience of BU compilation is that, to compile a CNF formula in L, one must first compile several of its subformulas. Empirical observations show that intermediate subformulas sometimes require representations that are much larger than both the input and the output of the compiler [10, 16]. A good BU compiler tries to avoid such hard subformulas by combining the clauses in a smart way following heuristics. Unfortunately, for some formulas, hard subformulas simply are unavoidable. This has been proved for several models of BU compilation into OBDD [14, 23, 9, 12] and more generally into strDNNF [8, 13]. To be more precise, these results show exponential lower bounds on the size of intermediate results for L-based *refutations* of *unsatisfiable* CNF formulas [1], which can be seen as BU compilations to L when they do not use weakening or projection rules.

In this paper, we study how BU compilers differ space-wise for different strategies to combine the clauses. Our main result is that the variant of BU compilation often used in practice, which we call *incremental* BU compilation, is less powerful than general BU compilation in the sense that for some formulas, incremental BU compilation always generates intermediate representations of exponential size, while general BU compilation can avoid it. Let us now precise what we mean by *incremental* BU compilers for CNF formulas. These are compilers that work clause by clause. An incremental compiler keeps in memory a single representation for subformulas – that we call the *core* – and repeatedly combines the core with a clause until it represents the whole formula. So whenever an “Apply” is done, one of the inputs is the core and the other represents a clause. Importantly, the next clause to conjoin to the core may be selected just before the “Apply”, so the order for combining the clauses is not necessarily decided ahead of compilation. Examples of incremental compiler can be found in [16] for the compilation of configuration problems into BDD; in the compiler SALADD[1] for compiling systems of pseudo-Boolean constraints into multi-valued decision diagrams (MDD); and we will later argue that the approach described in [3], which is the basis for the default compiler of the SDD package[2], can in fact be simulated by incremental compilation with only a polynomial size increase. The BU framework makes no assumption on how the constraints or clauses are combined and our objective is to show that fixing a strategy, while completely legitimate in practice, results in unavoidable exponential-size intermediate representations for some formulas that are otherwise “easy” to compile. Formally our result is the following

▶ **Theorem 1.** *There is an infinite class Φ of CNF formulas such that every $\phi \in \Phi$ over n variables admits polynomial-size compilations in OBDD($\wedge$,r) but all its incremental compilations in strDNNF($\wedge$,r) create intermediate circuits of size $2^{\Omega(\sqrt{n})}poly(1/n)$.*

An $L(\wedge, r)$ compilation refers to a BU compilation in L that uses the “Apply” to conjoin elements in L that are “similarly structured” and where arbitrary modifications preserving equivalence can be made to an intermediate result before it is fed to an “Apply” (this accounts for the so-called “restructuring” or “reordering” operation). To put our result into perspective, for $L \in \{$OBDD, SDD, strDNNF$\}$, the lower bounds shown in [14, 23, 12, 8, 13] hold regardless of the BU compilation strategy and, in the few cases where the clauses are chosen in a certain order, for instance in [9], it is not clear that the formulas for the lower bounds are easily compiled non-incrementally. Further, we are not aware of practical BU compilers for CNF formulas that are not incremental or that can not be simulated by incremental compilation. We do not claim that the development of non-incremental

[1] https://www.irit.fr/~Helene.Fargier/BR4CP/CompilateurSALADD.html
[2] http://reasoning.cs.ucla.edu/sdd/

compilers is the path to follow since we doubt of the practicability of that strategy. But we argue that while lower bounds in the general general framework are strong results, positive results on the other hand have no practical implication unless they can be proved for the incremental strategy. Things are different for OBDD-based refutations that rely on the weakening rule, so we can not make any claim in this context. Some refutation strategies improve upon the incremental approach using a clustering step [17, 18]. These approaches are non-incremental but are not very relevant to this work since the clusters are defined so that variables can be forgotten (i.e., existentially quantified out) after compilation of a cluster, which is forbidden in our setting as this modifies the function to compile. This is also the reason why the techniques used by Segerlind to separate tree-like OBDD-based refutations and general OBDD-based refutations [22] are not applicable in our setting, despite the apparent proximity with our topic.

The paper is organized as follows. We start in Section 2 with preliminaries where we describe the compilation languages considered and the general framework for BU compilation. Then in Section 3 we formalize incremental BU compilation and discuss some of its advantages compared to general BU compilation. The main part of the paper is Section 4 where we prove our main result on the separation between incremental and general BU compilation. Finally, Section 5 briefly discusses the implications of this result in practice and regarding the choice of a framework for modeling the behavior of algorithms.

2 Preliminaries

A Boolean variable is a variable x over $\{0,1\}$ (0 for *false*, 1 for *true*). An assignment to a set X of Boolean variables is a mapping from X to $\{0,1\}$. We call $\{0,1\}^X$ the set of assignments to X. A Boolean function f over X is a mapping from $\{0,1\}^X$ to $\{0,1\}$. When not specified, $var(f)$ denotes the set of variables of f. A literal is Boolean variable x or its negation $\overline{x}$. We use the usual symbols $\wedge$ and $\vee$ for conjunction and disjunction. A *clause* is a disjunction of literals and a *CNF formula* ϕ is a conjunction of clauses. We often see ϕ as the set of its clauses, so that we can write $c \in \phi$ to denote that c is a clause of ϕ, and $\phi \setminus c$ to denote the CNF formula whose clauses are all clauses of ϕ except c. Given a set of clauses S, we sometimes write $\bigwedge S$ to denote the CNF formula $\bigwedge_{c \in S} c$. CNF formulas are representations of Boolean functions. Two representations Σ and Σ' are *equivalent*, denoted by $\Sigma \equiv \Sigma'$ if they represent the same function, in particular they must be defined over the same set of variables. We say that Σ *entails* Σ', denoted by $\Sigma \models \Sigma'$ if $var(\Sigma') \subseteq var(\Sigma)$ and every assignment to $var(\Sigma)$ that satisfies Σ also satisfies Σ'.

Circuits in strDNNF

A circuit is in negation normal form (NNF) is a Boolean circuit whose gates are $\vee$-gates and $\wedge$-gates and whose inputs are literals or $\{0,1\}$ inputs. In particular there are no $\neg$-gates in a circuit in NNF. The set of variables below a gate g is denoted by $var(g)$. A gate g with inputs $g_1, \dots, g_k$ is called *decomposable* when $var(g_i) \cap var(g_j) = \emptyset$ for every $i \neq j$. A circuit in NNF is in decomposable NNF (DNNF) when all its $\wedge$-gates are decomposable. Circuits in strDNNF respect a more constrained variant of decomposability called *structured decomposability.* It requires a *vtree* (variable tree), that is, a rooted binary tree T whose leaves are in bijection with the circuit's variables. For t a node of T, we denote by $var(t)$ the set of variables that label leaves under t. The circuit D in DNNF *respects* T when all $\wedge$-gates have fan-in 2 and when there is a mapping λ from D's gates to T's nodes such that for every gate g of D,

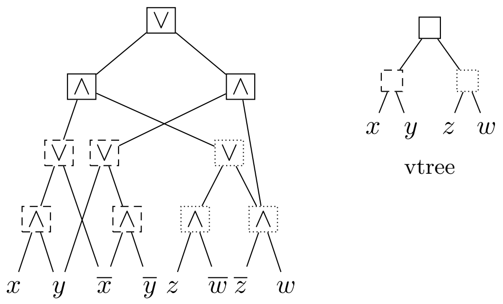
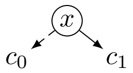


Figure 1 A circuit in strDNNF.

- $var(g) \subseteq var(\lambda(g))$,
- if g is a $\vee$-gate with inputs $g_1, \dots, g_k$, then $\lambda(g) = \lambda(g_1) = \dots = \lambda(g_k)$,
- if g is a $\wedge$-gate with inputs g_1, g_2 then $\lambda(g) = t$ is an internal node of T and there is a node t_1 under its first child and a node t_2 under its second child such that $\lambda(g_1) = t_1$ and $\lambda(g_2) = t_2$.

The size of D, denoted by $|D|$ is the number of connectors in the circuit. Not all circuits in DNNF are in strDNNF but the class of circuits in strDNNF, called the *strDNNF language*, is sufficiently expressive to represent all Boolean functions over finitely many variables [19]. An example of circuit in strDNNF is shown Figure 1: the mapping between internal gates and nodes of the vtree is represented with solid, dashed and dotted boxes. Important function representations in compilation admit linear-time translations into strDNNF, including SDDs whose definition we omit (see [4]) and OBDDs which we define now.

OBDDs

Given two Boolean functions c_0, c_1 and a Boolean variable x, the *decision node* 

represents the function $(\overline{x} \wedge c_0) \vee (x \wedge c_1)$. Graphically, if x is set to 0 then follow the dashed arrow, otherwise if x is set to 1 then follow the solid arrow. A binary decision diagram (BDD) is a directed acyclic graph (DAG) with a single root and two sinks labelled by 0 and 1, and whose internal nodes are decision nodes. BDDs are interpreted as Boolean functions over the variables that label their decision nodes. The satisfying assignments are those whose unique corresponding path leads to the sink 1. An *ordered BDD* (OBDD) is such that the variables appear in the same order and at most once along every path from the root to a sink. Examples of OBDDs are shown in Example 2. The class of OBDDs is called the *OBDD language.* The size of an OBDD B, denoted by $|B|$, is its number of nodes. OBDDs can be transformed in linear time into strDNNF circuits respecting *linear vtrees*, that is, vtrees where every internal node has a child that is a leaf. So OBDDs can be seen as being structured by linear vtrees.

Bottom-up compilation

Given a language L whose elements are structured by vtrees and a system of finitely many constraints ϕ (for instance a CNF formula, where constraints are clauses), an $L(\wedge, r)$ *bottom-up compilation* of ϕ in L is a sequence

$$(\Sigma_1, I_1), (\Sigma_2, I_2), \dots, (\Sigma_N, I_N)$$

where $\Sigma_1, \dots, \Sigma_N$ are elements of L such that $\Sigma_N \equiv \phi$ and where, for every $i \in \{1, \dots, N\}$, I_i is an instruction telling how Σ_i is obtained. Three types of instructions are possible:

- $\Sigma_i = \mathsf{Compile}(c)$ for some constraint c of ϕ. Then $\Sigma_i \equiv c$.
- $\Sigma_i = \mathsf{Apply}(\Sigma_j, \Sigma_k, \wedge)$ for some $j, k < i$ such that $\Sigma_i, \Sigma_j, \Sigma_k$ respect the *same* vtree. Then $\Sigma_i \equiv \Sigma_j \wedge \Sigma_k$.
- $\Sigma_i = \mathsf{Restructure}(\Sigma_j)$ for some $j < i$. Then $\Sigma_i \equiv \Sigma_j$ and their vtrees may differ.

The *size* of the compilation is defined as $\max_{1 \leq i \leq N} |\Sigma_i|$. The Restructure instruction accounts for the r in the name "$L(\wedge, r)$ compilation". When this instruction is not used, we say that we have an $L(\wedge)$ compilation. Regarding the Apply instruction, the assumption that Σ_i, Σ_j and Σ_k respect the same vtree is essential since, for many languages L, polynomial-time conjunction of two elements in L into a another element in L is feasible only when the initial elements have compatible vtrees. Quadratic-time Apply procedures for conjunction are known for the languages considered in this paper, namely for OBDD [2], SDD [4] and strDNNF [19].

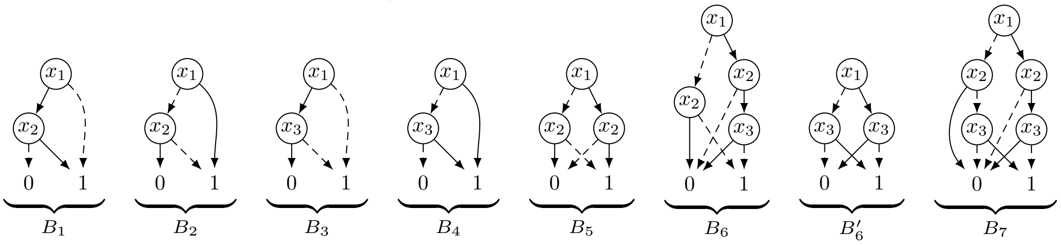
3 Incremental and Non-Incremental Bottom-Up Compilation

To compile a CNF formula in a bottom-up manner, some compilers work clause by clause. The idea is to keep one main representation, which we call the *core*, to which clauses are conjoined one after the other. We call this strategy *incremental* bottom-up compilation. Formally, an incremental $L(\wedge, r)$ compilation is an $L(\wedge, r)$ compilation where every Apply instruction is of the form $\mathsf{Apply}(\Sigma, \mathsf{Compile}(c), \wedge)$, where $\Sigma \in L$ has been computed previously in the compilation and c is a clause/constraint of the formula to compile. We used a simplified framework where, for all I_i, the core is Σ_{i-1} (by convention $\Sigma_0 = 1$).

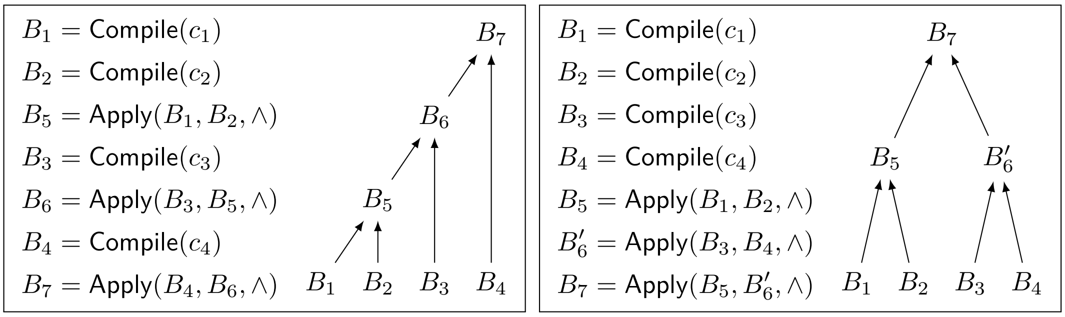
- $\Sigma_i = \mathsf{Apply}(\Sigma_{i-1}, \Sigma_c, \wedge)$ where $c \in \phi$ and $\Sigma_c \equiv c$ respects the *same* vtree as Σ_{i-1} and Σ_i.
- $\Sigma_i = \mathsf{Restructure}(\Sigma_{i-1})$. Then $\Sigma_i \equiv \Sigma_{i-1}$ and their vtrees may differ.

For convenience the compilation of Σ_c is implicit. We can visualize the compilation $(\Sigma_1, I_1), \dots, (\Sigma_N, I_N)$ as a directed acyclic graph whose vertices are in bijection with the Σ_is. If I_i is a Compile instruction, then Σ_i is a source of the DAG, else if I_i is $\Sigma_i = \mathsf{Restructure}(\Sigma_j)$ then there is an edge from Σ_j to Σ_i, and if I_i is $\Sigma_i = \mathsf{Apply}(\Sigma_j,\ \Sigma_k,\ \wedge)$, then there is an edge from Σ_j to Σ_i and another one from Σ_k to Σ_i. In this DAG representation, if we bypass the nodes of in-degree 1 created by the Restructure instructions, then an incremental BU compilation is shaped like a *linear* tree, that is, a tree such where every node is either a leaf or has a leaf child. In general BU compilation, the DAG can be shaped like a tree, in which case we have a *tree-like* compilation, or it can have a general DAG structure (some Σ_is may be inputs to more than one Apply).

▶ **Example 2.** Consider the CNF formula $\phi = (\overline{x}_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (x_1 \vee x_3) = c_1 \wedge c_2 \wedge c_3 \wedge c_4$, and the following OBDDs:



B_1, B_2, B_3 and B_4 represent the clauses c_1, c_2, c_3 and c_4, respectively. Now consider the following bottom-up compilations of ϕ:



$B_1 = \mathsf{Compile}(c_1)$
$B_2 = \mathsf{Compile}(c_2)$
$B_5 = \mathsf{Apply}(B_1, B_2, \wedge)$
$B_3 = \mathsf{Compile}(c_3)$
$B_6 = \mathsf{Apply}(B_3, B_5, \wedge)$
$B_4 = \mathsf{Compile}(c_4)$
$B_7 = \mathsf{Apply}(B_4, B_6, \wedge)$

$B_1 = \mathsf{Compile}(c_1)$
$B_2 = \mathsf{Compile}(c_2)$
$B_3 = \mathsf{Compile}(c_3)$
$B_4 = \mathsf{Compile}(c_4)$
$B_5 = \mathsf{Apply}(B_1, B_2, \wedge)$
$B'_6 = \mathsf{Apply}(B_3, B_4, \wedge)$
$B_7 = \mathsf{Apply}(B_5, B'_6, \wedge)$

The left compilation is incremental whereas the one on the right is tree-like but non-incremental since B_7 is obtained from B_5 and B'_6, none of which represents a clause from ϕ.

In incremental BU compilation, the constraints are conjoined in a certain order. The framework is general enough that, in practice, the order for the constraints is not necessarily fixed before the first Apply: an incremental BU compiler that has run a few steps and computed the core Σ can dynamically decide the next clause to conjoin based on Σ.

Some strategies for BU compilation are definitely not incremental but can be simulated by incremental BU compilation. For instance, we claim it is the case of the compiler scheme described in [3] for compiling a CNF formula into an SDD. Since we have omitted the definition of SDDs, we look at it as a compilation to strDNNF (of which SDD is a sublanguage). Say $\phi(X)$ be the CNF formula to compile and let T be a vtree over X. Let T_v denotes the subtree of T rooted under node v. Reusing some of the terminology in [4], we say that each clause c of ϕ is *hosted* at the unique deepest node v of T such that $var(c) \subseteq var(v)$. Let ϕ_v be the subformula of ϕ made uniquely of clauses over $var(v)$ and let ϕ_v^{host} be the subformula of ϕ made uniquely of clauses of ϕ hosted at v. Suppose v is not a leaf and let v_1 and v_2 be its children. The idea is, given T, to recursively compute a strDNNF circuit $D_v \equiv \phi_v$ for the node v as follows

(1) compute the strDNNF circuits $D_{v_1} \equiv \phi_{v_1}$ and $D_{v_2} \equiv \phi_{v_2}$ for v_1 and v_2 respecting the vtrees T_{v_1} and T_{v_2} respectively
(2) compute $D = \mathsf{Apply}(D_{v_1}, D_{v_2}, \wedge)$ that respects T_v
(3) incrementally conjoin all clauses of ϕ_v^{host} to D, perhaps restructuring D to change its vtree under v between two Apply (so T is modified but only under v).

The strDNNF circuit for the root node of T represents ϕ. Observe that, by definition of a vtree, in step (2) there is $var(\phi_{v_1}) \cap var(\phi_{v_2}) = var(v_1) \cap var(v_2) = \emptyset$, so we say that the Apply occurring there is *decomposable*. We claim that this approach can be simulated by an incremental strDNNF($\wedge$,r) compilation because any Apply that does not fit the incremental approach is a *decomposable* Apply from step (2) and such Apply are easy to simulate in the incremental approach. The proof of the following is deferred to the appendix.

▶ **Proposition 3.** *Every strDNNF($\wedge$,r) compilation of a CNF formula ϕ where every* Apply *that does not involve a clause of ϕ computes the conjunction of two strDNNF circuits representing subformulas of ϕ over disjoint set of variables, can be transformed into an incremental strDNNF($\wedge$,r) compilation with only a polynomial increase in size.*

Let us mention some advantages of incremental BU compilation. First, many guarantees on the size of the final output of a BU compilation also hold for incremental BU compilation. Especially, it is known that if the primal graph of a CNF formula has logarithmic treewidth, then the formula can be compiled into a polynomial-size SDD [4] and, since the primal treewidth cannot increase for subformulas it follows that, in the case of a logarithmic treewidth, polynomial-size compilations in SDD($\wedge$,r) exist even in the incremental case.

Another example is on the role of restructuring in BU compilation: it is shown in [12] that OBDD($\wedge$,r) compilations are exponentially more compact than OBDD($\wedge$) compilations and a careful observation of the proof reveals that the OBDD($\wedge$,r) compilations used there can be made incremental. So restructuring does not require stronger strategies than incremental compilation to have a positive effect. From a practical point of view, incremental compilers have the advantage that the purpose of restructuring is clear (at least when compiling CNF formulas): it is to reduce the size of the core. This is fairly intuitive and corresponds to what is done in practice, in particular by the default compiler of the SDD package. One can actually prove that there is not much interest space-wise in using restructuring for anything else as any incremental $L(\wedge,r)$ compilation for $L \in \{\text{OBDD}, \text{SDD}, \text{strDNNF}\}$ can be transformed in polynomial-time into another $L(\wedge,r)$ compilation where restructuring never increases the size of the core. The proof of the following appears in appendix.

▶ **Proposition 4.** *Let $L \in \{OBDD, SDD, strDNNF\}$, let $(\Sigma_1, I_1), \dots, (\Sigma_N, I_N)$ be an incremental $L(\wedge,r)$ compilation of the CNF formula ϕ over n variables. There exists an incremental $L(\wedge,r)$ compilation $(\Sigma'_1, I'_1), \dots, (\Sigma'_M, I'_M)$ of ϕ such that $M \leq 2N$ and, for every $j \in \{1, \dots, M\}$ there is an $i \in \{1, \dots, N\}$ such that $\Sigma'_j \equiv \Sigma_i$ and $|\Sigma'_j| \leq 2n|\Sigma_i|$. In addition, when Σ'_j is obtained by restructuring Σ'_{j-1}, we have $|\Sigma'_j| \leq |\Sigma'_{j-1}|$.*

We note that Proposition 4 holds because every clause has a linear-size representation in L for every vtree. The statement holds for other types of constraints that share this property, for instance parity constraints [20], but it is not clear whether it applies to general systems of constraints as well.

In non-incremental BU compilation, restructuring may not be used only for reducing the size. Indeed, given two circuits in L respecting different vtrees over the same variables, using an Apply to obtain their conjunction requires changing the vtree of at least one of them, so it requires a restructuring step. But it can be the case that there is no way to make both circuits respect the same vtree without a global size increase.

4 Separating Incremental and Non-Incremental Compilation

In this section, we prove the main result of the paper, which is the separation between incremental and non-incremental bottom-up compilation.

▶ **Theorem 1.** *There is an infinite class Φ of CNF formulas such that every $\phi \in \Phi$ over n variables admits polynomial-size compilations in OBDD($\wedge$,r) but all its incremental compilations in strDNNF($\wedge$,r) create intermediate circuits of size $2^{\Omega(\sqrt{n})}poly(1/n)$.*

4.1 General Idea for the Separation

First we sketch the general idea for proving the separation. We use two CNF formulas $\phi_1(X)$ and $\phi_2(X)$ and consider the formula $\phi(X) = \bigwedge_{c_1 \in \phi_1} \bigwedge_{c_2 \in \phi_2} (c_1 \vee c_2)$. This formula is in CNF and is equivalent to $\phi_1 \vee \phi_2$. Let us say that ϕ_1 is unsatisfiable, then ϕ is equivalent to ϕ_2. To compile ϕ in non-incremental OBDD($\wedge$,r), we can first compile independently the subformulas $\bigwedge_{c_1 \in \phi_1} (c_1 \vee c_2) \equiv \phi_1 \vee c_2 \equiv c_2$ for every $c_2 \in \phi_2$. This gives one OBDD B_{c_2} for every clause of ϕ_2. Any clause can be represented by a small OBDD with respect to any variable ordering so, using restructuring, we can freely choose the variable ordering for B_{c_2}. Then, starting from the OBDDs B_{c_2} for all $c_2 \in \phi_2$, we can compile an OBDD representing ϕ as if we were doing a BU compilation of ϕ_2. The idea is summarized in Figure 2

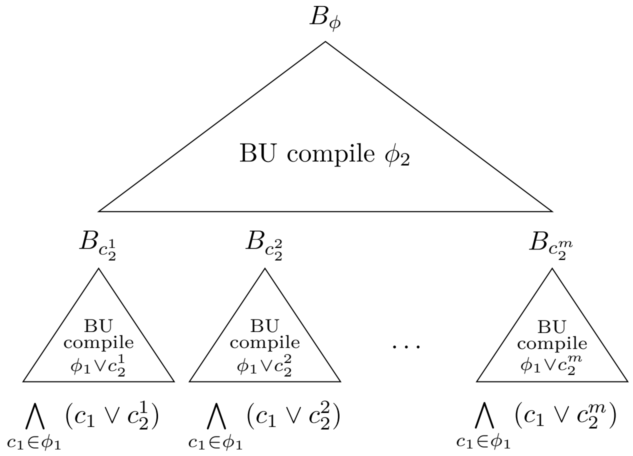

B_ϕ
BU compile ϕ_2
$B_{c_2^1}$
$B_{c_2^2}$
$B_{c_2^m}$
BU compile $\phi_1 \vee c_2^1$
BU compile $\phi_1 \vee c_2^2$
...
BU compile $\phi_1 \vee c_2^m$
$\bigwedge_{c_1 \in \phi_1} (c_1 \vee c_2^1)$
$\bigwedge_{c_1 \in \phi_1} (c_1 \vee c_2^2)$
$\bigwedge_{c_1 \in \phi_1} (c_1 \vee c_2^m)$

Figure 2 A bottom-up compilation of $\phi \equiv \phi_1 \vee \phi_2$ when ϕ_1 is unsatisfiable.

where $c_2^1, c_2^2, \ldots, c_2^m$ are the clauses of ϕ_2. The small triangles represent BU compilations of $\bigwedge_{c_1 \in \phi_1}(c_1 \vee c_2^i)$. Each costs roughly the same as a BU compilation of ϕ_1. So the cost of this compilation scheme is roughly m times the cost of a BU compilation of ϕ_1, plus the cost of a BU compilation of ϕ_2.

Now what happens when compiling ϕ incrementally in strDNNF($\wedge$,r) (which is more general than OBDD($\wedge$,r))? Well, let us look at the last instruction Apply of the compilation: suppose it is $D' = \mathsf{Apply}(D, D_{\gamma_1 \vee \gamma_2}, \wedge)$, that is, we conjoin the core circuit D with a circuit in strDNNF representing the clause $\gamma_1 \vee \gamma_2$ where $\gamma_1 \in \phi_1$ and $\gamma_2 \in \phi_2$. But then D computes the CNF formula $\phi \setminus (\gamma_1 \vee \gamma_2)$, which one can show is equivalent to $(\phi_2 \vee (\phi_1 \setminus \gamma_1)) \wedge (\phi_1 \vee (\phi_2 \setminus \gamma_2)) \equiv ((\phi_1 \setminus \gamma_1) \wedge (\phi_2 \setminus \gamma_2)) \vee \phi_2$ since ϕ_1 is unsatisfiable. For convenience let us assume that ϕ_2 is also unsatisfiable and that the minimal unsatisfiable subsets of ϕ_1 and ϕ_2 are themselves (so both $\phi_1 \setminus \gamma_1$ and $\phi_2 \setminus \gamma_2$ are satisfiable). Then the cost of compiling ϕ incrementally is at least the cost of compiling $(\phi_1 \setminus \gamma_1) \wedge (\phi_2 \setminus \gamma_2)$.

To separate general BU compilation from incremental BU compilation, we look for formulas ϕ_1 and ϕ_2 that are both easy to compile in OBDD($\wedge$,r), so that ϕ is easy to compile in non-incremental OBDD($\wedge$,r) using the scheme of Figure 2, but such that $(\phi_1 \setminus \gamma_1) \wedge (\phi_2 \setminus \gamma_2)$ can only be represented by exponential-size strDNNF circuits regardless of the vtree and regardless of the choice of γ_1 and γ_2. In the next sections, we describe our formulas and give the formal proof. For the sake of keeping the proof simple, we will choose a formula ϕ_2 that is satisfiable and we will not necessarily look at the very last Apply of the compilation, but the idea behind the proof is unchanged.

4.2 The Formula ϕ for the Separation

Let $X = \{x_{i,j} \mid 1 \leq i, j \leq n\}$ and consider the following functions:

- ROW$_n(X)$ (resp. COL$_n(X)$) is the function over X whose satisfying assignments are that for which each row (resp. column) of the $n \times n$ matrix $[x_{i,j}]_{i,j}$ contains exactly one 1.
- ODD(X) (resp. EVEN(X)) is the function over X whose satisfying assignments are that for which there is an odd (resp. even) number of 1s in the $n \times n$ matrix $[x_{i,j}]_{i,j}$.

We will consider CNF formulas *representing* ROW$_n(X)$ and COL$_n(X)$, and CNF formulas *encoding* ODD(X) and EVEN(X). We distinguish a representation from an encoding in that the latter uses auxiliary variables while the former does not. The CNF formulas representing ROW$_n(X)$ and COL$_n(X)$ are:

$$row_n(X) = \bigwedge_{1\leq i\leq n} \Big((x_{i,1} \vee \cdots \vee x_{i,n}) \wedge \bigwedge_{1\leq j<k\leq n} (\overline{x}_{i,j} \vee \overline{x}_{i,k}) \Big)$$

$$col_n(X) = \bigwedge_{1\leq j\leq n} \Big((x_{1,j} \vee \cdots \vee x_{n,j}) \wedge \bigwedge_{1\leq h<i\leq n} (\overline{x}_{h,j} \vee \overline{x}_{i,j}) \Big)$$

ODD and EVEN do not have polynomial-size representations in CNF but they have linear-size encodings. We use the notation $(a+b+c=0 \mod 2)$ to denote the clauses $(a \vee \overline{b} \vee c) \wedge (a \vee b \vee \overline{c}) \wedge (\overline{a} \vee b \vee c) \wedge (\overline{a} \vee \overline{b} \vee \overline{c})$. We use the standard encoding [21]:

$$odd_n(X,Y) = y_n \wedge \bigwedge_{1\leq j\leq n} \bigwedge_{1\leq i\leq n} (y_{i,j} + y_{i-1,j} + x_{i,j} = 0 \mod 2)$$

where $y_{0,j} = y_{n,j-1}$ if $j > 1$, and $y_{0,1} = 0$. In $odd_n(X,Y)$, the matrix is browsed row-wise and a satisfying assignment sets $y_{i,j}$ to 1 if and only if the number of 1s found before the cell i,j is odd.

We define $even_n(X,Y)$ analogously: we just replace the $(a+b+c=0 \mod 2)$ by $(a+b+c=1 \mod 2)$, which clauses are $(a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee \overline{c}) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (a \vee b \vee c)$. The following holds:

$$row_n(X) \equiv \text{ROW}_n(X) \text{ and } col_n(X) \equiv \text{COL}_n(X) \text{ and}$$
$$\exists Y.odd_n(X,Y) \equiv \text{ODD}(X) \text{ and } \exists Y.even_n(X,Y) \equiv \text{EVEN}(X).$$

Let n be an *even* integer, let $\phi_1(X) = row_n(X)$, and let

$$\phi_1^*(X,Y) = \phi_1(X) \wedge odd_n(X,Y) \quad \text{and} \quad \phi_2(X) = \bigwedge_{1\leq j\leq n} \bigwedge_{1\leq h<i\leq n} (\overline{x}_{h,j} \vee \overline{x}_{i,j}). \tag{1}$$

ϕ_2 represents the function that accepts exactly the matrices $[x_{i,j}]_{i,j}$ whose columns all contains at most one 1. We will later refer to this function as AMOpCOL_n (at most one per column). ϕ_1^* is unsatisfiable since n is even. We will use the following CNF formula for the separation:

$$\phi = \bigwedge_{c_1\in\phi_1^*} \bigwedge_{c_2\in\phi_2} (c_1 \vee c_2) \equiv \phi_1^* \vee \phi_2 \equiv \phi_2. \tag{2}$$

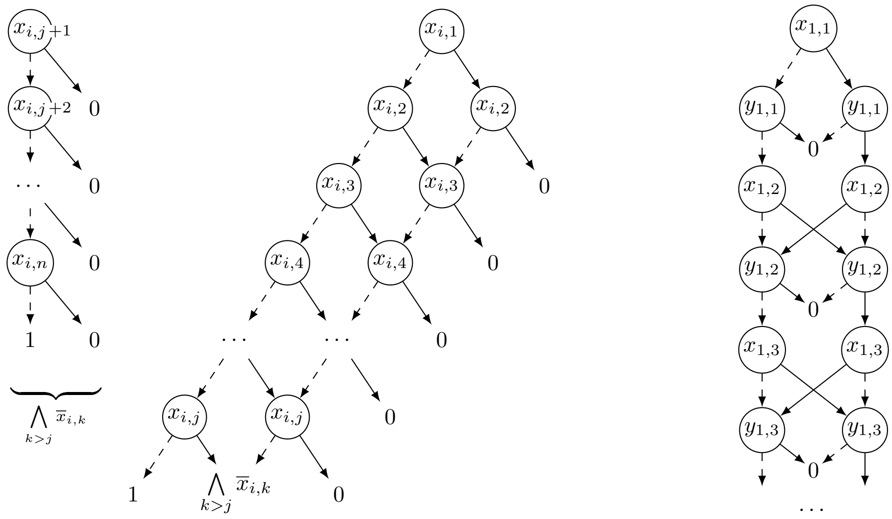
4.3 ϕ is Easy to Compile in OBDD($\wedge$,r)

Let us start with the proof that ϕ can be compiled in OBDD($\wedge$,r) using only polynomial-size OBDDs. The key element here is that both ϕ_1^* and ϕ_2 are easily compiled in OBDD($\wedge$,r).

▶ **Lemma 5.** *The formulas ϕ_1^* and ϕ_2 defined in (1) have polynomial-size OBDD($\wedge$,r) compilations.*

Proof. Let us start with ϕ_1^*. We fix an i between 1 and n. For every j, we separately compile $\bigwedge_{k>j}(\overline{x}_{i,j} \vee \overline{x}_{i,k}) \equiv \overline{x}_{i,j} \vee \bigwedge_{k>j} \overline{x}_{i,k}$ into an OBDD B_j that respects the variable ordering $x_{i,1}, x_{i,2}, \dots, x_{i,n}$. It is easy to see that compiling each B_j is feasible in polynomial time in OBDD($\wedge$,r).

▷ Claim 6. $B_1 \wedge \cdots \wedge B_j$ is equivalent to the OBDD represented in Figure (3a).



(a) Intermediate step in the compilation of row_n. **(b)** Intermediate step in the compilation of odd_n.

Figure 3 Intermediate steps in the compilation of ϕ_1^*.

The proof of the Claim 6 is deferred to the appendix. We compile $B_1 \wedge \cdots \wedge B_n \equiv \bigwedge_{j\neq k}(\overline{x}_{i,j} \vee \overline{x}_{i,k})$ using only OBDDs of size $O(n)$. Let B^i be the resulting OBDD. Next, it is easy to compile the clause $x_{i,1} \wedge \cdots \wedge x_{i,n}$ into an OBDD of size $O(n)$ and that respects the variable ordering $x_{i,1}, x_{i,2}, \ldots, x_{i,n}$. We conjoin this OBDD to B^i and thus obtain an OBDD that represents $x_{i,1} + \cdots + x_{i,n} = 1$ whose size is $O(n)$ and that we call $B_{\text{ith } row}$. Now the OBDDs $B_{\text{ith } row}$ for every $1 \leq i \leq n$ have disjoint sets of variables so they can be incrementally conjoined into a single OBDD B_{rows} of size $O(n^2)$ that represents ROW$_n$ and whose variable ordering is $x_{1,1}, \ldots, x_{1,n}, x_{2,1}, \ldots, x_{2,n}, \ldots, x_{n,n}$.

It remains to compile odd_n and to conjoin it to B_{rows}. We can compile odd_n incrementally in OBDD($\wedge$,r) using only the variable ordering $x_{1,1}, y_{1,1}, x_{1,2}, y_{1,2}, \ldots, x_{1,n}, y_{1,n}, x_{2,1}, y_{2,1}, \ldots, \quad x_{n,n}, y_{n,n}$. First we conjoin the OBDDs for the clauses of the parity constraint $y_{1,1} + x_{1,1} = 0 \mod 2$ together, then we add the OBDDs for the clauses of the parity constraint $y_{1,1} + x_{1,2} + y_{1,2} = 0 \mod 2$, and so on. Once we have added all clauses of a parity constraint, we end up with an OBDD that looks like that represented Figure (3b). To add the next parity constraints, we only have four OBDDs to conjoin before we attain again an OBDD that looks like that represented Figure (3b). So the sizes of the OBDDs used to compile odd_n never grow bigger than $O(n^2)$. We call B_{odd} the final OBDD.

Finally, since B_{odd} and B_{rows} have compatible variable orderings, they can be conjoined in quadratic time, and the compilation of ϕ_1^* is finished.

For ϕ_2, the OBDD($\wedge$,r) compilation scheme used to compile the OBDD B^i computing $\bigwedge_{j\neq k}(\overline{x}_{i,j} \vee \overline{x}_{i,k})$, respecting the variable ordering $x_{i,1}, \ldots, x_{i,n}$, and whose is size is in $O(n)$, can be adapted to compile an OBDD Q^j computing $\bigwedge_{h\neq i}(\overline{x}_{h,j} \vee \overline{x}_{i,j})$, respecting the variable ordering $x_{1,j}, \ldots, x_{n,j}$ and whose size is in $O(n)$. Since the OBDDs $Q^1, \ldots, Q^n$ have pairwise disjoint sets of variables, they can be incrementally conjoined in polynomial time to obtain an OBDD computing $Q^1 \wedge \cdots \wedge Q^n \equiv \phi_2$. ◀

▶ **Lemma 7.** *The formula ϕ defined by (2) can be compiled in OBDD($\wedge$,r) using only OBDDs of size polynomial in n.*

Proof. For every clause $c_2 \in \phi_2$, we use Lemma 5 to construct a small OBDD($\wedge$,r) compilation of $\bigwedge_{c_1 \in \phi_1^*}(c_1 \vee c_2)$: we take the polynomial-size OBDD($\wedge$,r) compilation of ϕ_1^* described in the lemma and we apply a disjunction between every OBDD of the compilation and a small OBDD representing c_2. This gives a compilation of $\bigwedge_{c_1 \in \phi_1^*}(c_1 \vee c_2) \equiv \phi_1^* \vee c_2 \equiv c_2$ and increases the size of every OBDD by a constant factor only. We do this for each clause $c_2 \in \phi_2$ independently and obtain an OBDD B_{c_2} for every c_2. Then we use restructuring to change the variable ordering of the B_{c_2}s (clauses admit linear-size OBDDs under every variable ordering) so that we can use the polynomial-size compilation of ϕ_2 described in Lemma 5. ◀

4.4 ϕ is Hard to Compile Incrementally in strDNNF($\wedge$,r)

We move to the more difficult part that consists in proving that every incremental strDNNF($\wedge$,r) compilation of ϕ generates intermediate circuits of exponential size. To this end, we will need the following lemma (which we prove later):

▶ **Lemma 8.** *Let ϕ be the formula defined by (2). Every incremental strDNNF($\wedge$,r) compilation of ϕ generates an intermediate circuit that can be transformed in polynomial-time into a circuit in strDNNF computing $ROW_{n-\Delta} \wedge COL_{n-\Delta}$ for some integer $\Delta \leq 3$.*

The function $\mathrm{ROW}_n(X) \wedge \mathrm{COL}_n(X)$ is widely studied in computer sciences and is often called the *permanent*. Its satisfying assignments are the 0/1 matrices $[x_{i,j}]_{i,j}$ that contain exactly one 1 in every row and exactly one 1 in every column. In particular, these assignments are in bijection with the permutations of $\{1, \dots, n\}$ since every assignment a to X that satisfies $\mathrm{ROW}_n(X) \wedge \mathrm{COL}_n(X)$ uniquely corresponds to the permutation that maps every $i \in \{1, \dots, n\}$ to the unique $j \in \{1, \dots, n\}$ such that $a(x_{i,j}) = 1$. The permanent is known to be hard to represent as OBDDs, as read-once branching programs [25, Theorem 6.2.12.][3], as circuits in strDNNF [24, Proposition 7 and Lemma 27], and even as circuits in DNNF.

▶ **Lemma 9** ([7, Proof of Theorem 1]). *Every circuit in DNNF representing $ROW_n(X) \wedge COL_n(X)$ has size $2^{\Omega(n)}$.*

The desired lower bound on incremental bottom-up compilation of ϕ follows directly from Lemmas 8 and 9. Thus the proof of Theorem 1 is straightforward given Lemmas 7, 8 and 9. The rest of the section is dedicated to the proof of Lemma 8. Due to space constraint, the proof of several claims is deferred to appendix.

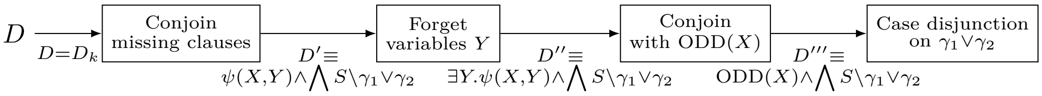
Consider an incremental strDNNF($\wedge$, r) compilation $(D_1, I_1), \dots, (D_N, I_N)$ of ϕ. The Apply operations are of the form

$$D_{i+1} = \mathsf{Apply}(D_i, D_{c_1 \vee c_2}, \wedge).$$

where $c_1 \in \phi_1 \wedge odd_n$ and $c_2 \in \phi_2$. Consider the *largest* integer $k < N$ such that

- we have that $D_{k+1} = \mathsf{Apply}(D_k, D_{\gamma_1 \vee \gamma_2}, \wedge)$
- and γ_1 is a clause of $\phi_1 = row_n$ (and not a clause of odd_n)
- and there are no clauses $c_1 \in \phi_1$ and $c_2 \in \phi_2$ such that $c_1 \vee c_2 \neq \gamma_1 \vee \gamma_2$ and $c_1 \vee c_2 \models \gamma_1 \vee \gamma_2$.

[3] The functions that we call ROW_n and COL_n do not equal the functions ROW_n and COL_n from [25]. Our $\mathrm{ROW}_n(X) \wedge \mathrm{COL}_n(X)$ corresponds to PERM_n in [25]



D
$D{=}D_k$
Conjoin missing clauses
$D'{\equiv}$
$\psi(X,Y)\wedge\bigwedge S\backslash\gamma_1\vee\gamma_2$
Forget variables Y
$D''{\equiv}$
$\exists Y.\psi(X,Y)\wedge\bigwedge S\backslash\gamma_1\vee\gamma_2$
Conjoin with ODD(X)
$D'''{\equiv}$
$\text{ODD}(X)\wedge\bigwedge S\backslash\gamma_1\vee\gamma_2$
Case disjunction on $\gamma_1\vee\gamma_2$

Figure 4 Strategy for proving Lemma 8.

For this k we have that every D_h for $h > k+1$ is obtained either by a Restructure operation, or by an Apply with an strDNNF circuit $D_{c_1 \vee c_2}$ such that c_1 is a clause of odd_n, or by an Apply with an strDNNF circuit $D_{c_1 \vee c_2}$ that is entailed by D_{k+1}.

▷ Claim 10. For any $h > k+1$ such that $D_h = \mathsf{Apply}(D_{h-1}, D_{c_1 \vee c_2}, \wedge)$ and $c_1 \in row_n$, we have that $D_k \models c_1 \vee c_2$ or $\gamma_1 \vee \gamma_2 \models c_1 \vee c_2$.

In the rest of the proof, we write $D = D_k$ for convenience. The strategy for showing Lemma 8 is to turn D into a circuit in strDNNF representing $\text{ROW}_{n-\Delta} \wedge \text{COL}_{n-\Delta}$ by means of a few polynomial-time transformations, as schematized in Figure 4. This differs slightly from the strategy described in Section 4.1: we are not considering the very last Apply. The reason is that the formula ϕ contains some clauses c such that $\phi \setminus c \models c$. A few of these clauses are even tautological. In a sense, these clauses are "irrelevant" in ϕ. Clearly, if the last Apply of the compilation is conjoining the core to an irrelevant clause, then this Apply is in fact not modifying the core and we should look at a previous Apply. One could think that we could prove the statement for a formula ϕ where irrelevant clauses have been removed. This is certainly true if we only remove tautologuous clauses. However we do not want to remove the irrelevant clauses that are non-tautologous since such clauses are not necessarily irrelevant for subformulas of ϕ. So, rather than removing irrelevant clauses from ϕ, we choose to keep them and to consider another Apply than the very last one.

By definition, not all clauses of the form $c_1 \vee c_2$ with $c_1 \in \phi_1$ and $c_2 \in \phi_2$ have been conjoined to the core during the first k steps of the compilation. Let $S = \{c_1 \vee c_2 \mid c_1 \in \phi_1, c_2 \in \phi_2\}$ and let $S_{D\models} = \{c \in S \mid D \models c\}$ be the set of clauses of S that are entailed by D and let $S' = S_{D\models} \setminus \{\gamma_1 \vee \gamma_2\}$. Since $\gamma_1 \vee \gamma_2$ has not been conjoined to D we have that

$$D \equiv \psi \wedge \bigwedge S'$$

where ψ is a subformula of $\bigwedge_{c \in odd_n} \bigwedge_{c_2 \in \phi_2} (c \vee c_2)$. Ideally we would like S' to be $S \setminus \{\gamma_1 \vee \gamma_2\}$, but this is generally not the case. Given a clause c, let $S_{c\models} = \{c' \mid c' \in S, c' \neq c \text{ and } c \models c'\}$ be the set of clauses in S that are distinct from c and that are entailed by c. By Claim 10, each clause in S is either entailed by D or is entailed by $\gamma_1 \vee \gamma_2$, so $S' \cup S_{\gamma_1 \vee \gamma_2 \models} = S \setminus \{\gamma_1 \vee \gamma_2\}$.

▷ Claim 11. Let $\gamma = \gamma_1 \vee \gamma_2$. For every vtree over X, there is a strDNNF of size $O(n^2)$ respecting that vtree that represents $\bigwedge S_{\gamma\models}$ (by convention $\bigwedge \emptyset = 1$).

By Claim 11, there is an strDNNF circuit $D_{\gamma_1 \vee \gamma_2 \models}$ of size $O(n^2)$ that computes $\bigwedge S_{\gamma_1 \vee \gamma_2 \models}$, and that respects the same vtree as D, so let $D' = \mathsf{Apply}(D, D_{\gamma_1 \vee \gamma_2 \models}, \wedge)$ whose size is $O(n^2|D|)$. We have that:

$$\begin{aligned} D' &\equiv \psi \wedge \bigwedge (S' \cup S_{\gamma_1 \vee \gamma_2 \models}) = \psi \wedge \bigwedge (S \setminus \{\gamma_1 \vee \gamma_2\}) \\ &\equiv \psi \wedge \bigwedge_{c_1 \in \phi_1 \setminus \gamma_1} \bigwedge_{c_2 \in \phi_2} (c_1 \vee c_2) \wedge \bigwedge_{c_2 \in \phi_2 \setminus \gamma_2} \bigwedge_{c_1 \in \phi_1} (c_1 \vee c_2) \\ &\equiv \psi \wedge ((\phi_1 \setminus \gamma_1) \vee \phi_2) \wedge (\phi_1 \vee (\phi_2 \setminus \gamma_2)) \end{aligned}$$

The next step is to get rid of ψ, or rather to replace it somehow by the function $\text{ODD}(X)$ that is more convenient to manipulate.

▷ Claim 12. We have that $\text{ODD}(X) \models \exists Y.\psi(X, Y)$.

In knowledge compilation, the *forgetting* transformation is, given a Boolean function representation Σ in some fixed language L and a set $Z \subset \mathit{var}(\Sigma)$, to modify Σ into another function representation Σ' in L that is equivalent to $\exists Z.\Sigma$ [6]. Forgetting is feasible in linear time for the language of circuits in strDNNF [19]. So we can construct a circuit D'' in strDNNF representing $\exists Y.D'(X,Y)$ and such that $|D''| = O(|D'|) = O(n^2|D|)$. Since ϕ_1 and ϕ_2 are formulas over X, we have that

$$D''(X) \equiv \exists Y.D'(X,Y) \equiv ((\phi_1 \setminus \gamma_1) \vee \phi_2) \wedge (\phi_1 \vee (\phi_2 \setminus \gamma_2)) \wedge \exists Y.\psi(X,Y).$$

For every vtree over X, there exists a strDNNF circuit that represents $\mathrm{ODD}(X)$, that respects the vtree, and whose size is in $O(|X|)$ [20, Proposition 5]. Let D_{ODD} be such a circuit respecting the same vtree as D''. Then we have $D''' = \mathsf{Apply}(D'', D_{\mathrm{ODD}}, \wedge)$ whose size is $O(|X||D''|) = O(n^4|D|)$ and such that

$$D'''(X) \equiv D''(X) \wedge \mathrm{ODD}(X) \equiv \big((\phi_1 \setminus \gamma_1) \vee \phi_2\big) \wedge \big(\phi_1 \vee (\phi_2 \setminus \gamma_2)\big) \wedge \mathrm{ODD}(X),$$

where the last equivalence follows from Claim 12. Furthermore, since $\mathrm{ODD}(X) \wedge \phi_1(X) = \mathrm{ODD}(X) \wedge \mathit{row}_n(X)$ is unsatisfiable (because n is even), we have that

$$D'''(X) \equiv \Big(\big((\phi_1 \setminus \gamma_1) \wedge (\phi_2 \setminus \gamma_2)\big) \vee \phi_2\Big) \wedge \mathrm{ODD}(X).$$

We have gotten rid of ψ as promised. The rest of the proof requires a case disjunction over γ_1 and γ_2.

- If $\gamma_1 = (x_{i,1} \vee \cdots \vee x_{i,n})$ and $\gamma_2 = (\overline{x}_{i',j} \vee \overline{x}_{i'',j})$, we can assume, without loss of generality, that $i \neq i'$ and $i \neq i''$ for otherwise $\gamma_1 \vee \gamma_2$ would be tautological. We consider the partial assignment a to X that maps all variables of the ith row to 0, that maps $x_{i',j}$ and $x_{i'',j}$ to 1, and all other variable of the i'th and i''th row 0, and that maps two columns distinct from the jth column to 0. We denote by X' all other variables left unassigned. For instance when $i = 1$, $i' = 2$, $i'' = 3$ and $j = 1$ we may have:

  $$a = \left(\begin{array}{ccc|ccc|} 0 & 0 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & 0 & \cdots & \cdots & 0 \\ 1 & 0 & 0 & \cdots & \cdots & 0 \\ \cline{4-6} 0 & 0 & 0 & & & \\ \vdots & \vdots & \vdots & & X' & \\ 0 & 0 & 0 & & & \\ \cline{4-6} \end{array}\right).$$

  Then we observe that $(\phi_1 \setminus \gamma_1)|a = (\mathit{row}_n \setminus \gamma_1)|a = \mathit{row}_{n-3}(X')$, that $\phi_2|a \equiv 0$, that $(\phi_2 \setminus \gamma_2)|a \equiv \mathrm{AMOpCOL}_{n-3}(X')$, and that $\mathrm{ODD}(X)|a = \mathrm{ODD}(X')$. Thus

  $$D'''(X)|a \equiv \mathit{row}_{n-3}(X') \wedge \mathrm{AMOpCOL}_{n-3}(X') \wedge \mathrm{ODD}(X'),$$

  but since $n-3$ is odd, we have that $\mathit{row}_{n-3}(X') \models \mathrm{ODD}(X')$ and thus

  $$D'''(X)|a \equiv \mathit{row}_{n-3}(X') \wedge \mathrm{AMOpCOL}_{n-3}(X') \equiv \mathrm{ROW}_{n-3}(X') \wedge \mathrm{COL}_{n-3}(X').$$

- If $\gamma_1 = (\overline{x}_{i,j} \vee \overline{x}_{i,j'})$ and $\gamma_2 = (\overline{x}_{i,j} \vee \overline{x}_{i',j})$, then we consider the partial assignment a to X that maps $x_{i,j}$, $x_{i,j'}$ and $x_{i',j}$ to 1, and all other variables of the ith and i'th rows to 0, and all other variables of the jth and j'th column to 0. We denote by X' all other variables left unassigned. For instance when $i = 1$, $i' = 2$, $j = 1$ and $j' = 2$ we have:

$$a = \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \multicolumn{3}{|c|}{} \\ \vdots & \vdots & \multicolumn{3}{|c|}{X'} \\ 0 & 0 & \multicolumn{3}{|c|}{} \end{pmatrix}.$$

Then we observe that $(\phi_1 \setminus \gamma_1)|a = (row_n \setminus \gamma_1)|a = row_{n-2}(X')$, that $\phi_2|a \equiv 0$, that $(\phi_2 \setminus \gamma_2)|a \equiv \mathrm{AMOpCOL}_{n-2}(X')$, and that $\mathrm{ODD}(X)|a = \mathrm{EVEN}(X')$. Thus

$$D'''(X)|a \equiv row_{n-2}(X') \wedge \mathrm{AMOpCOL}_{n-2}(X') \wedge \mathrm{EVEN}(X'),$$

but since $n-2$ is even, we have that $row_{n-2}(X') \models \mathrm{EVEN}(X')$ so

$$D'''(X)|a \equiv row_{n-2}(X') \wedge \mathrm{AMOpCOL}_{n-2}(X') \equiv \mathrm{ROW}_{n-2}(X') \wedge \mathrm{COL}_{n-2}(X').$$

- If $\gamma_1 = (\overline{x}_{i,j} \vee \overline{x}_{i,j'})$ and $\gamma_2 = (\overline{x}_{i',j''} \vee \overline{x}_{i'',j''})$, where j j' and j'' are pairwise distinct and where i, i' and i'' are pairwise distinct, then we consider the partial assignment a to X that maps $x_{i,j}$, $x_{i,j'}$, $x_{i',j''}$ and $x_{i'',j''}$ to 1, and that maps all other variables of the ith, i'th and i''th rows to 0, and that maps all other variables of the jth, j'th and j''th columns to 0. We denote by X' all other variables left unassigned. For instance when $i = 1$, $i' = 2$, $i'' = 3$ and $j = 2$, $j' = 3$, $j'' = 1$ we have:

$$a = \begin{pmatrix} 0 & 1 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \multicolumn{3}{|c|}{} \\ \vdots & \vdots & \vdots & \multicolumn{3}{|c|}{X'} \\ 0 & 0 & 0 & \multicolumn{3}{|c|}{} \end{pmatrix}.$$

Then we observe that $(\phi_1 \setminus \gamma_1)|a = (row_n \setminus \gamma_1)|a = row_{n-3}(X')$, that $\phi_2|a \equiv 0$, that $(\phi_2 \setminus \gamma_2)|a \equiv \mathrm{AMOpCOL}_{n-3}(X')$, and that $\mathrm{ODD}(X)|a = \mathrm{ODD}(X')$. Thus

$$D'''(X)|a \equiv row_{n-3}(X') \wedge \mathrm{AMOpCOL}_{n-3}(X') \wedge \mathrm{ODD}(X'),$$

but since $n-3$ is odd, we have that $row_{n-3}(X') \models \mathrm{ODD}(X')$ and thus

$$D'''(X)|a \equiv row_{n-3}(X') \wedge \mathrm{AMOpCOL}_{n-3}(X') \equiv \mathrm{ROW}_{n-3}(X') \wedge \mathrm{COL}_{n-3}(X').$$

All other cases are either similar to the these three, or correspond to cases where $\gamma_1 \vee \gamma_2$ is tautological or entailed by some other clause of S, which is impossible due to the assumptions on k at the beginning of the proof. So, in all relevant cases, we have a partial assignment a to X such that

$$D'''(X)|a \equiv \mathrm{ROW}_{n-\Delta}(X') \wedge \mathrm{COL}_{n-\Delta}(X')$$

for some $\Delta \leq 3$. Then we just condition D''' on a in linear time and we obtain a strDNNF circuit over X' that computes $\mathrm{ROW}_{n-\Delta}(X') \wedge \mathrm{COL}_{n-\Delta}(X')$. This concludes the proof of Lemma 8. Given Lemmas 7 and 9, Theorem 1 follows.

5 Discussion and Conclusion

In this paper we have shown that non-incremental bottom-up (BU) compilation largely outperforms incremental BU compilation space-wise. This raises several questions which we address here. Perhaps the first that comes to mind is: what does it mean regarding potential improvements of practical BU compilers? We have mentioned a non-incremental BU compilation algorithm in Proposition 3 and the discussion before but only to give an example that could be simulated by incremental BU compilation. Looking back at Figure 2, it seems clustering is the key to the efficient compilation of ϕ into OBDD: first the clauses are partitioned into disjoint clusters, then each cluster is compiled, and finally the resulting circuits/diagrams are compiled incrementally together. Clustering is used in OBDD-based refutations where heuristics create each cluster so that a variable can be forgotten (i.e., existentially quantified out) after compilation of the cluster [17, 18]. Contrary to refutations, compilations do not allow forgetting variables so one should think of different clustering strategies. But it is reasonable to believe that some formula can actually be compiled quite efficiently incrementally while a systematic clustering step into a constant number of clusters would make the compilation much harder. Actually it is not clear this work will fuel any practical research on BU compilation. Indeed it is not hard to buy into the claim that hypothetical efficient non-incremental BU compilers ought to be much more complex that incremental ones. Aguably, finding a good ordering of the clauses for incremental compilers is not an easy task and it is reasonable to expect that finding a good "DAG ordering" or "tree ordering" is much harder as the space of possible orderings gets larger.

A discussion that we think is more interesting is about the strength of the frameworks used to analyze practical algorithms. The framework for BU compilation is very general and thus exponential lower bounds in this framework are strong results. On the other hand, positive results may not translate into practical observations as they are obtained in a framework that is too general. In our case, the positive results on the BU compilation of the formulas from Theorem 1 will not be observed for any of the practical BU compilers that we are aware of, as they all work (close to) incrementally. Going further, narrow frameworks can also help explaining the behavior of algorithms on certain instances. For instance in [8], specific CNF formulas are shown to be hard for BU compilation and the proof distinguishes two cases: one where the last Apply conjoins representations for two large subformulas, and another one where it conjoins at least one representation of a subformula with only a few clauses. To explain the behavior of incremental BU compilers on these formulas, the second case would be enough, and this case is arguably the easiest one in [8]. A future direction for us is the comparison of top-down (TD) compilers and BU compilers. Are there formulas that are provably hard for TD compilers (described in a suitable framework) but easy for BU compilers, and vice-versa? For the separation to be observed empirically, tractability results for BU compilation should use the incremental approach. In particular, should the formulas of Theorem 1 be hard for TD compilers, we would not be completely satisfied saying that these formulas answer one direction of the problem.

From a purely theoretical perspective, there are still open problems on BU compilation. One that is related to this work is the separation of tree-like and general BU compilation. A careful observation of the nice BU compilations of the formulas of Theorem 1 shows that they are tree-like, that is, every intermediate OBDD is an input to at most one Apply. We ask whether there exists a class of formulas that have polynomial-size BU compilation but such that all tree-like BU compilations generate intermediate results of exponential size.

References

1 Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint Propagation as a Proof System. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 – October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004. doi:10.1007/978-3-540-30201-8_9.

2 Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–692, 1986.

3 Arthur Choi and Adnan Darwiche. Dynamic Minimization of Sentential Decision Diagrams. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6470.

4 A. Darwiche. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *Proc. of IJCAI'11*, pages 819–826, 2011.

5 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001. doi:10.1145/502090.502091.

6 Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi:10.1613/jair.989.

7 Alexis de Colnet. A Lower Bound on DNNF Encodings of Pseudo-Boolean Constraints. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 2020. doi:10.1007/978-3-030-51825-7_22.

8 Alexis de Colnet and Stefan Mengel. Lower Bounds on Intermediate Results in Bottom-Up Knowledge Compilation. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 – March 1, 2022*, pages 5564–5572. AAAI Press, 2022. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20496.

9 Luke Friedman and Yixin Xu. Exponential Lower Bounds for Refuting Random Formulas Using Ordered Binary Decision Diagrams. In Andrei A. Bulatov and Arseny M. Shur, editors, *Computer Science – Theory and Applications – 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings*, volume 7913 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2013. doi:10.1007/978-3-642-38536-0_11.

10 Jinbo Huang and Adnan Darwiche. Using DPLL for Efficient OBDD Construction. In *SAT 2004 – The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004. URL: http://www.satisfiability.org/SAT04/programme/61.pdf.

11 Jinbo Huang and Adnan Darwiche. DPLL with a Trace: From SAT to Knowledge Compilation. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 – August 5, 2005*, pages 156–162. Professional Book Center, 2005. URL: http://ijcai.org/Proceedings/05/Papers/0876.pdf.

12 Dmitry Itsykson, Alexander Knop, Andrei E. Romashchenko, and Dmitry Sokolov. On OBDD-based Algorithms and Proof Systems that Dynamically Change the order of Variables. *J. Symb. Log.*, 85(2):632–670, 2020. doi:10.1017/jsl.2019.53.

13 Dmitry Itsykson, Artur Riazanov, and Petr Smirnov. Tight Bounds for Tseitin Formulas. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 6:1–6:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAT.2022.6.

14 Jan Krajícek. An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams. *J. Symb. Log.*, 73(1):227–237, 2008. doi:10.2178/jsl/1208358751.

15 Pierre Marquis. Compile! In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 4112–4118. AAAI Press, 2015. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9596.

16 Nina Narodytska and Toby Walsh. Constraint and Variable Ordering Heuristics for Compiling Configuration Problems. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 149–154, 2007. URL: http://ijcai.org/Proceedings/07/Papers/022.pdf.

17 Guoqiang Pan and Moshe Y. Vardi. Search vs. Symbolic Techniques in Satisfiability Solving. In *SAT 2004 – The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, 2004. URL: http://www.satisfiability.org/SAT04/programme/62.pdf.

18 Guoqiang Pan and Moshe Y. Vardi. Symbolic Techniques in Satisfiability Solving. *J. Autom. Reason.*, 35(1-3):25–50, 2005. doi:10.1007/s10817-005-9009-7.

19 Knot Pipatsrisawat and Adnan Darwiche. New Compilation Languages Based on Structured Decomposability. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 517–522. AAAI Press, 2008. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-082.php.

20 Knot Pipatsrisawat and Adnan Darwiche. Top-Down Algorithms for Constructing Structured DNNF: Theoretical and Practical Implications. In Helder Coelho, Rudi Studer, and Michael J. Wooldridge, editors, *ECAI 2010 – 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 3–8. IOS Press, 2010. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=17704.

21 Steven D. Prestwich. CNF Encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 75–100. IOS Press, 2021. doi:10.3233/FAIA200985.

22 Nathan Segerlind. On the Relative Efficiency of Resolution-Like Proofs and Ordered Binary Decision Diagram Proofs. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity, CCC 2008, 23-26 June 2008, College Park, Maryland, USA*, pages 100–111. IEEE Computer Society, 2008. doi:10.1109/CCC.2008.34.

23 Olga Tveretina, Carsten Sinz, and Hans Zantema. Ordered Binary Decision Diagrams, Pigeonhole Formulas and Beyond. *J. Satisf. Boolean Model. Comput.*, 7(1):35–58, 2010. doi:10.3233/sat190074.

24 Romain Wallon and Stefan Mengel. Revisiting Graph Width Measures for CNF-Encodings. *J. Artif. Intell. Res.*, 67:409–436, 2020. doi:10.1613/jair.1.11750.

25 Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000. URL: http://ls2-www.cs.uni-dortmund.de/monographs/bdd/.

A Appendix

A.1 Missing Proofs of Section 3

We show the following lemma before proving Proposition 3.

▶ **Lemma 13.** *Let ϕ_1 and ϕ_2 be such that $var(\phi_1) \cap var(\phi_2) = \emptyset$. Given $(D^1_1, I^1_1), \dots, (D^1_s, I^1_s)$ an incremental strDNNF(∧,r) compilation of ϕ_1, and $(D^2_1, I^2_1), \dots, (D^2_t, I^2_t)$ an incremental strDNNF(∧,r) compilation of ϕ_2, there is an incremental strDNNF(∧,r) compilation of $\phi_1 \wedge \phi_2$ whose largest elements has size at most $\max_i |D^1_i| + \max_j |D^2_j| + 1$.*

Proof. Just take the compilation $(D_1, I_1), \dots, (D_s, I_s), (D_{s+1}, I_{s+1}), \dots, (D_{s+t}, I_{s+t})$ where $I_i = I_i^1$ and $D_i = D_i^1$ for $1 \leq i \leq s$ and I_{s+j} is I_j where every D_j^2 is replaced by $D_{s+j} = D_s \wedge D_j^2$ (that is, the root of D_{s+j} is one $\wedge$-node whose children are D_s and D_j^2). Since $var(D_s) \cap var(D_j^2) \subseteq var(\phi_1) \cap var(\phi_2) = \emptyset$, each D_{s+j} is a strDNNF circuit. It is a bottom-up compilation of $\phi_1 \wedge \phi_2$ whose size is at most $\max(\max_i |D_i^1|, 1 + |D_s^1| + \max_j |D_j^2|)$. ◀

▶ **Proposition 3.** *Every strDNNF($\wedge$,r) compilation of a CNF formula ϕ where every* Apply *that does not involve a clause of ϕ computes the conjunction of two strDNNF circuits representing subformulas of ϕ over disjoint set of variables, can be transformed into an incremental strDNNF($\wedge$,r) compilation with only a polynomial increase in size.*

Proof. We can restrict our focus to BU compilations where all circuits constructed, except the last one, are used as premises for some Restructure or Apply instructions. For D an intermediate circuit in the compilation, let $pred(D)$ be the set containing D plus all circuits constructed before D in the compilation and that are used in the construction of D. More formally, if $D = \mathsf{Compile(C)}$ then $pred(D) = \{D\}$, if $D = \mathsf{Apply}(D', D'', \wedge)$ then $pred(D) = \{D\} \cup pred(D') \cup pred(D'')$ and if $D = \mathsf{Restructure}(D')$ then $pred(D) = \{D\} \cup pred(D')$. In particular, calling $D^* \equiv \phi$ the final circuit of the compilation, $pred(D^*)$ is the set of all circuits generated during the compilation. Let further k_D be the number of decomposable Apply done to construct D. Consider the last decomposable Apply in the compilation: $D = \mathsf{Apply}(D', D'', \wedge)$ with $var(D') \cap var(D'') = \emptyset$. Suppose there is an incremental strDNNF($\wedge$,r) compilation of D' (resp. D'') whose largest circuit has size at most $k_{D'} + \sum_{K \in pred(D')} |K|$ (resp. $k_{D''} + \sum_{K \in pred(D'')} |K|$). By Lemma 13, there is an incremental BU compilation of D that generates only circuits of size at most $1 + k_{D'} + k_{D''} + \sum_{K \in pred(D')} |K| + \sum_{K \in pred(D'')} |K|$. And since $pred(D')$ and $pred(D'')$ are disjoint for a decomposable Apply, this upper bound equals $k_D + \sum_{K \in pred(D)} |K|$. Calling $D^* \equiv \phi$ the final circuit of the compilation, we obtain by induction that ϕ can be compiled by an incremental strDNNF($\wedge$,r) compilation that generates only circuits of size at most $k_{D^*} + \sum_{K \in pred(D^*)} |K|$ where $k_{D^*} = O(|\phi|)$, which is a polynomial size increase compared to the original compilation. ◀

We are now going to prove Proposition 4. Before that, we recall a seemingly insignificant property of OBDDs, SDDs and strDNNF circuits: given a set X of variables, for every vtree T over X, every clause c over X has a representation in OBDD, SDD and strDNNF that respects T and whose size is $O(|c|)$.

▶ **Proposition 4.** *Let $L \in \{OBDD, SDD, strDNNF\}$, let $(\Sigma_1, I_1), \dots, (\Sigma_N, I_N)$ be an incremental L($\wedge$,r) compilation of the CNF formula ϕ over n variables. There exists an incremental L($\wedge$,r) compilation $(\Sigma'_1, I'_1), \dots, (\Sigma'_M, I'_M)$ of ϕ such that $M \leq 2N$ and, for every $j \in \{1, \dots, M\}$ there is an $i \in \{1, \dots, N\}$ such that $\Sigma'_j \equiv \Sigma_i$ and $|\Sigma'_j| \leq 2n|\Sigma_i|$. In addition, when Σ'_j is obtained by restructuring Σ'_{j-1}, we have $|\Sigma'_j| \leq |\Sigma'_{j-1}|$.*

Proof. We follow the instructions $I_1, \dots, I_N$ and construct $(\Sigma'_1, I'_1), \dots, (\Sigma'_M, I'_M)$ along the way. We denote by S_i the subsequence of this sequence, that we have after obtaining Σ_i. We begin with $\Sigma'_1 = \Sigma_1$, $I'_1 = I_1$ and $S_1 = ((\Sigma'_1, I'_1))$. We will denote by $S_i \cdot x$ the sequence obtained adding x at the end of S_i. We say that invariant (I) holds at step i when, after Σ_i is constructed, denoting $S_i = ((\Sigma'_1, I'_1), \dots, (\Sigma'_j, I'_j))$, we have that $\Sigma'_j \equiv \Sigma_i$ and $|\Sigma'_j| \leq |\Sigma_i|$. (I) clearly holds at step 1. Now let us assume that (I) holds at step i, let $j = |S_i|$, and consider the instruction I_{i+1}. There are three cases to consider.

- I_{i+1} is $\Sigma_{i+1} = \mathsf{Restructure}(\Sigma_i)$. If $|\Sigma_{i+1}| > |\Sigma'_j|$ then, since $\Sigma_{i+1} \equiv \Sigma_i \equiv \Sigma'_j$, we simply define $S_{i+1} = S_i$. Otherwise, if $|\Sigma_{i+1}| \leq |\Sigma'_j|$ then we define $\Sigma'_{j+1} = \Sigma_{i+1}$ and $I'_{j+1} : \Sigma'_{j+1} = \mathsf{Restructure}(\Sigma'_j)$ (recall that $\Sigma'_j \equiv \Sigma_i \equiv \Sigma_{i+1} = \Sigma'_{j+1}$ by the invariant) and $S_{i+1} = S_i \cdot (\Sigma'_{j+1}, I'_{j+1})$. In both cases, invariant (I) holds at step $i + 1$.

- I_{i+1} is $\Sigma_{i+1} = \mathsf{Apply}(\Sigma_i, \Sigma_c, \wedge)$ where $c \in \phi$. Let Σ'_c be the *linear-size* representation of c in L that respects the same vtree as Σ'_j. Now define I'_{j+1} as $\Sigma'_{j+1} = \mathsf{Apply}(\Sigma'_j, \Sigma'_c, \wedge)$. We have that $\Sigma'_j \equiv \Sigma_i$ so $\Sigma'_{j+1} \equiv \Sigma_{i+1}$. Moreover, since $|\Sigma'_c| \leq 2|var(c)| \leq 2n$ and since the Apply is a quadratic-time procedure, we have that $|\Sigma'_{j+1}| \leq 2n|\Sigma'_j| \leq 2n|\Sigma_i|$. If $|\Sigma'_{j+1}| \leq |\Sigma_{i+1}|$, then we define $S_{i+1} = S_i \cdot (\Sigma'_{j+1}, I'_{j+1})$. Otherwise if $|\Sigma'_{j+1}| > |\Sigma_{i+1}|$, then we define $\Sigma'_{j+2} = \Sigma_{i+1}$, $I'_{j+2} : \Sigma'_{j+2} = \mathsf{Restructure}(\Sigma'_{j+1})$ and $S_{i+1} = S_i \cdot (\Sigma'_{j+1}, I'_{j+1}) \cdot (\Sigma'_{j+2}, I'_{j+2})$. In both cases, invariant (I) holds at step $i+1$.

At the end of the construction we have $S_N = ((\Sigma'_1, I'_1), \dots, (\Sigma'_M, I'_M))$ and $\Sigma'_M \equiv \Sigma_N \equiv \phi$ since (I) holds at step N. At every step, we add at most two elements to S_i, so $M \leq 2N$. Furthermore, for every $j \geq 1$, Σ'_{j+1} is either the result of an Apply between Σ'_j and a clause of ϕ, or comes from restructuring Σ'_j. So S_N is an incremental $L(\wedge,r)$ compilation of ϕ. Looking back at our construction of S_i, we see that (1) restructuring is only used when it decreases the size of the OBDD and (2) that every Σ'_j is equivalent to some Σ_i and that its size is never greater than $2n|\Sigma_i|$. ◀

A.2 Missing Proofs of Section 4

▷ Claim 6. $B_1 \wedge \dots \wedge B_j$ is equivalent to the OBDD represented in Figure (3a).

Proof. Recall that $B_j \equiv \overline{x}_{i,j} \vee \bigwedge_{k>j} \overline{x}_{i,k}$ where i is fixed. So $B_1 \wedge B_2 \wedge \dots \wedge B_j \equiv \left(\overline{x}_{i,1} \vee \bigwedge_{k>1} \overline{x}_{i,k}\right) \wedge \left(\overline{x}_{i,2} \vee \bigwedge_{k>2} \overline{x}_{i,k}\right) \wedge \dots \wedge \left(\overline{x}_{i,j} \vee \bigwedge_{k>j} \overline{x}_{i,k}\right)$. We argue by case disjunction on $x_{i,1} + \dots + x_{i,j}$. First, any assignment to $(x_{i,k})_k$ that satisfies $x_{i,1} + \dots + x_{i,j} = 0$ clearly satisfies $B_1 \wedge B_2 \wedge \dots \wedge B_j$. Second, any assignment that satisfies $x_{i,1} + \dots + x_{i,j} = 1$ satisfies $B_1 \wedge B_2 \wedge \dots \wedge B_j$ if and only if it satisfies $\bigwedge_{k>j} \overline{x}_{i,k}$. Finally, any assignment that satisfies $x_{i,1} + \dots + x_{i,j} > 1$ falsifies $B_1 \wedge B_2 \wedge \dots \wedge B_j$. Thus $B_1 \wedge B_2 \wedge \dots \wedge B_j$ is equivalent to

$$(x_{i,1} + \dots + x_{i,j} = 0) \vee ((x_{i,1} + \dots + x_{i,j} = 1) \wedge (x_{i,j+1} + \dots + x_{i,n} = 0))$$

It is clear that the OBDD represented in Figure (3b) computes the function written above. ◁

▷ Claim 10. For any $h > k+1$ such that $D_h = \mathsf{Apply}(D_{h-1}, D_{c_1 \vee c_2}, \wedge)$ and $c_1 \in row_n$, we have that $D_k \models c_1 \vee c_2$ or $\gamma_1 \vee \gamma_2 \models c_1 \vee c_2$.

Proof. Suppose there is an $h > k+1$ such that $D_h = \mathsf{Apply}(D_{h-1}, D_{c_1 \vee c_2}, \wedge)$, $c_1 \in row_n$, and $D_k \not\models c_1 \vee c_2$ and $\gamma_1 \vee \gamma_2 \not\models c_1 \vee c_2$. If there is no other clause of ϕ that entails $c_1 \vee c_2$ then this contradicts the fact that k is maximal. Otherwise, if $c_1 \vee c_2$ is entailed by another clause $c'_1 \vee c'_2$ of ϕ, then $c'_1 \in row_n(X)$ since $c_1 \vee c_2$ uses only variables of X while every clause of $odd_n(X,Y)$ features an encoding variable of Y. In addition, the integer h' such that $D_{h'} = \mathsf{Apply}(D_{h'-1}, D_{c'_1 \vee c'_2}, \wedge)$ is greater than $k+1$, for otherwise we would have either $D_k \models D_{h'} \models D_{c'_1 \vee c'_2} \models D_{c_1 \vee c_2} \models c_1 \vee c_2$ (for $h' < k+1$) or $\gamma_1 \vee \gamma_2 = c'_1 \vee c'_2 \models c_1 \vee c_2$ (for $h' = k+1$). So we repeat the argument with the clause $c'_1 \vee c'_2$ instead of $c_1 \vee c_2$ until reaching a contradiction. ◁

▷ Claim 12. We have that $\mathrm{ODD}(X) \models \exists Y.\psi(X,Y)$.

Proof. First note that we have $odd_n(X,Y) \models odd_n(X,Y) \vee \phi_2(X) \models \psi(X,Y)$ since ψ is a subformula of a formula equivalent to $odd_n(X,Y) \vee \phi_2(X)$. Since $\exists Y.odd_n(X,Y) \equiv \mathrm{ODD}(X)$, it only remains to explain that $\exists Y.odd_n(X,Y) \models \exists Y.\psi(X,Y)$. Let a_X be an assignment to X satisfying $\exists Y.odd_n(X,Y)$. Then there exists an assignment a_Y to Y such that $a_X \cup a_Y$ satisfies $odd_n(X,Y)$. But then $a_X \cup a_Y$ satisfies $\psi(X,Y)$, and thus a_X satisfies $\exists Y.\psi(X,Y)$. So $\exists Y.odd_n(X,Y) \models \exists Y.\psi(X,Y)$. ◁

▷ Claim 11. Let $\gamma = \gamma_1 \vee \gamma_2$. For every vtree over X, there is a strDNNF of size $O(n^2)$ respecting that vtree that represents $\bigwedge S_{\gamma\models}$ (by convention $\bigwedge \emptyset = 1$).

Proof. Some clauses of S are tautological, that is, they contain a variable x and its negation $\overline{x}$ and are thus equivalent to 1, let $S_{taut} \subset S$ be the set of such clauses. We now consider the different possibilities for $\gamma_1 \vee \gamma_2$.

- If $\gamma_1 = (x_{i,1} \vee \cdots \vee x_{i,n})$ and $\gamma_2 = (\overline{x}_{i,j} \vee \overline{x}_{i',j})$ then γ is tautological so $S_{\gamma\models} \subseteq S_{taut}$.
- If $\gamma_1 = (x_{i,1} \vee \cdots \vee x_{i,n})$ and $\gamma_2 = (\overline{x}_{i',j} \vee \overline{x}_{i'',j})$ where i, i' and i'' are pairwise distinct, then $S_{\gamma\models} \subseteq S_{taut}$.
- If $\gamma_1 = (\overline{x}_{i,j} \vee \overline{x}_{i,j'})$ and $\gamma_2 = (\overline{x}_{i',j''} \vee \overline{x}_{i'',j''})$ where i, i', i'' are pairwise distinct and where j, j', j'' are pairwise distinct. Then $S_{\gamma\models} \subseteq S_{taut}$.
- If $\gamma_1 = (\overline{x}_{i,j} \vee \overline{x}_{i,j'})$ and $\gamma_2 = (\overline{x}_{i',j} \vee \overline{x}_{i'',j})$ where i, i', i'' are pairwise distinct. Then $S_{\gamma\models} \subseteq S_{taut}$.
- If $\gamma_1 = (\overline{x}_{i,j} \vee \overline{x}_{i,j'})$ and $\gamma_2 = (\overline{x}_{i,j''} \vee \overline{x}_{i',j''})$ where j, j', j'' are pairwise distinct. Then $S_{\gamma\models} \subseteq S_{taut}$.
- If $\gamma_1 = (\overline{x}_{i,j} \vee \overline{x}_{i,j'})$ and $\gamma_2 = (\overline{x}_{i,j} \vee \overline{x}_{i',j})$, then $S_{\gamma\models}$ is the set of all clauses of the form $\gamma_1 \vee (\overline{x}_{i',j} \vee \overline{x}_{i'',j})$ for $i'' \neq i$ and $i'' \neq i'$, plus all clauses of the form $(\overline{x}_{i,j'} \vee \overline{x}_{i,j''}) \vee \gamma_2$ for $j'' \neq j$ and $j'' \neq j'$, plus all the tautological clauses.

The claim is trivially true when $S_{\gamma\models} \subseteq S_{taut}$, so we only have the last case to consider. For any given vtree, we want an small strDNNF circuit that computes $\bigwedge_{i''\notin\{i,i'\}}(\gamma_1 \vee \overline{x}_{i',j} \vee \overline{x}_{i'',j}) \wedge \bigwedge_{j''\notin\{j,j'\}}(\overline{x}_{i,j'} \vee \overline{x}_{i,j''} \vee \gamma_2) \equiv \left(\gamma_1 \vee \overline{x}_{i',j} \vee \bigwedge_{i''\notin\{i,i'\}} \overline{x}_{i'',j}\right) \wedge \left(\gamma_2 \vee \overline{x}_{i,j'} \vee \bigwedge_{j''\notin\{j,j'\}} \overline{x}_{i,j''}\right)$. It is readily verified that, for any vtree, there are strDNNF circuits of size $O(n)$ that computes $\bigwedge_{i''\notin\{i,i'\}} \overline{x}_{i'',j}$ and $\bigwedge_{j''\notin\{j,j'\}} \overline{x}_{i,j''}$ and that there are strDNNF circuits of size of size $O(1)$ computing $\gamma_1 \vee \overline{x}_{i',j}$ and $\gamma_2 \vee \overline{x}_{i,j'}$. So, for any vtree, we start from these strDNNF circuits and apply two disjunctions and one conjunction to obtain a strDNNF circuit of size $O(n^2)$ computing the desired function. ◁

IPASIR-UP: User Propagators for CDCL

Katalin Fazekas
TU Wien, Austria

Aina Niemetz
Stanford University, CA, USA

Mathias Preiner
Stanford University, CA, USA

Markus Kirchweger
TU Wien, Austria

Stefan Szeider
TU Wien, Austria

Armin Biere
Universität Freiburg, Germany

Abstract

Modern SAT solvers are frequently embedded as sub-reasoning engines into more complex tools for addressing problems beyond the Boolean satisfiability problem. Examples include solvers for Satisfiability Modulo Theories (SMT), combinatorial optimization, model enumeration and counting. In such use cases, the SAT solver is often able to provide relevant information beyond the satisfiability answer. Further, domain knowledge of the embedding system (e.g., symmetry properties or theory axioms) can be beneficial for the CDCL search, but cannot be efficiently represented in clausal form. In this paper, we propose a general interface to inspect and influence the internal behaviour of CDCL SAT solvers. Our goal is to capture the most essential functionalities that are sufficient to simplify and improve use cases that require a more fine-grained interaction with the SAT solver than provided via the standard IPASIR interface. For our experiments, we extend CaDiCaL with our interface and evaluate it on two representative use cases: enumerating graphs within the SAT modulo Symmetries framework (SMS), and as the main CDCL(T) SAT engine of the SMT solver cvc5.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases SAT, CDCL, Satisfiability Modulo Theories, Satisfiability Modulo Symmetries

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.8

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.8003683`
Software (Source Code): `https://github.com/markirch/sat-modulo-symmetries/`
Software (Source Code): `https://github.com/cvc5/cvc5`
Software (Source Code): `https://github.com/arminbiere/cadical`

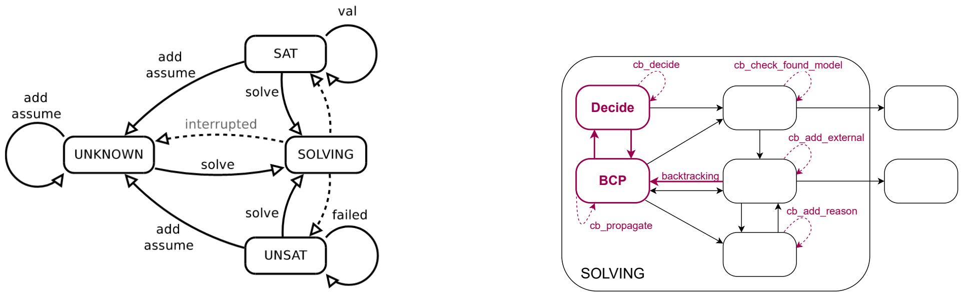
Funding This work was supported in part by the Stanford Center for Automated Reasoning, the Stanford Center for Blockchain Research, a gift from Amazon Web Services, by the Austrian Science Fund (FWF) under projects No. T-1306 and P-32441, and by the Vienna Science and Technology Fund (WWTF) under project No. ICT19-065 (Reveal-AI).

Acknowledgements The authors would like to thank the reviewers for their thoughtful comments and helpful suggestions on improving this paper.

1 Introduction

Modern SAT solvers frequently serve as crucial sub-reasoning engines of more complex tools for addressing problems beyond the Boolean satisfiability problem. Examples include solvers for Satisfiability Modulo Theories (SMT) [9], combinatorial problems [3,37], or model

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 8; pp. 8:1–8:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) The four possible states of SAT solvers according to the IPASIR interface (see [5]).

(b) The five additional states within state `Solving` according to the IPASIR-UP interface.

Figure 1 The IPASIR model and its extension with states and transitions within CDCL solving.

enumeration and counting [21]. The introduction of the IPASIR interface [5] enabled a relatively simple integration of off-the-shelf SAT solver as a black box into larger systems, typically to incrementally solve a sequence of similar propositional sub-problems.

Many use cases, however, require a tighter integration with a more fine-grained interaction of the SAT solver with the rest of the system. A prominent example is the CDCL($\mathcal{T}$) framework for SMT solvers [35], where the search of the core SAT solver on the propositional abstraction of the input problem is guided by theory solvers. Other use cases include MaxSAT solvers, which benefit from knowing if some literals imply others [22], and solvers for symmetric combinatorial problems, where it is desired to add additional clauses during search [15]. Currently, such use cases require either workarounds on the user level or non-trivial modifications of the SAT solver. As a consequence, it is non-trivial to replace the underlying SAT solver, which prevents taking advantage of recent advancements in SAT solving. Additionally, non-standard extensions and modifications of the SAT solver, if not done carefully, often come at the cost of an accidental performance hit.

In this paper, we propose a generic interface able to capture the essential functionalities necessary to simplify and improve such use cases of SAT solvers. For this purpose, we extend the IPASIR interface [5] with an interface to facilitate *external propagators*, also called *user propagators* (UP), yielding a new interface called IPASIR-UP.

Our extension allows users (1) to inspect and being notified about changes to the trail during search, (2) to add clauses to the problem during solving without restarting the search, and (3) to propagate literals directly, based on external knowledge, without explicitly adding reason clauses (i.e., using delayed on-demand explanation). Implementing support for such an interface is non-trivial in a state-of-the-art SAT solver, but enables a wide range of applications to efficiently use the solver without further, application-specific workarounds and modifications. To advocate our proposed interface, we implemented it in CaDiCaL [10], a state-of-the-art incremental SAT solver, on top of its implementation of the IPASIR interface.

Furthermore, we present two representative use cases of this extension of CaDiCaL in two different application contexts: integrating CaDiCaL via IPASIR-UP as the core SAT solver into (1) the CDCL-based SAT modulo Symmetries (SMS) framework and (2) a CDCL($\mathcal{T}$)-based Satisfiability Modulo Theories (SMT) solver. Our experiments present evidence that the IPASIR-UP interface provides a rich and concise interface for a modern, proof-producing, incremental SAT solver with inprocessing in such applications.

2 An Interface beyond IPASIR

The IPASIR interface, as introduced in [5], considers four possible states of a SAT solver (see Figure 1a). Initially, and while the formula is under construction, the solver is in state `Unknown`. When function `solve()` is called, it transitions into state `Solving`. From that state, the solver can transition to either `SAT` or `UNSAT` (or, on interruption, back to `Unknown`). Thus, IPASIR allows multiple calls to `solve()` while modifying the formula or querying details of the found solution (resp. refutation) *between* such calls. It is, however, not possible to interact with the solver while it is *in* the `Solving` state (except for interruptions). Our goal is to extend IPASIR with functions that can provide such interactions, and thereby allow to simplify and improve several use cases of modern incremental SAT solvers.

For this purpose, our interface IPASIR-UP refines the IPASIR state `Solving`, which implements the main CDCL loop, into *five* states, as shown in Figure 1b. CDCL combines unit propagation (`BCP`) with decisions (`Decide`) until either a clause becomes falsified by the current assignment or each variable is assigned a truth value. In the first case, the solver transitions into state `Conflict Analysis`, where it captures the reason of the contradiction as a derived driving clause, which is then learned in state `Learning`. If the learned clause is empty, the solver transitions to the `UNSAT` state. Otherwise, it backtracks to a lower decision level and unit propagation starts again. In the second case, as soon as a complete assignment is found, a standard CDCL solver will transition into the state `SAT`. In the presence of an *external propagator*, however, we introduce an artificial state called `Solution Analysis` as an intermediate state before transitioning to `SAT`.

In each of the five states in Figure 1b, IPASIR-UP provides a callback (with prefix "`cb_`") to interact with the external propagator (dashed transitions in Figure 1b, see Section 2.3). Additionally, the propagator is being notified about changes to the trail (states and solid transitions highlighted in purple in Figure 1b, see Section 2.2). In the following, we briefly describe the main purpose of each function. Though we illustrate IPASIR-UP here by an example implementation in C++ (see Listing 1 and Listing 2), the API is low-level enough to be supported in C as well. Note that many other (in this context) less relevant steps of the search (e.g. restart, reduce, and inprocessing) are ignored in the model of our interface.

2.1 Configuration and Management

In order to be able to interact with the solver while in the `Solving` state, a user may connect and configure an *external propagator* through IPASIR-UP as follows.

Setup. When the solver is not in the `Solving` state, the user can connect an external propagator via the function `connect_external_propagator`. This propagator may be disconnected outside of `Solving` via `disconnect_external_propagator`. There can be at most one external propagator connected to a solver.

Observed Variables. While an external propagator is connected, at any point in time (even during `Solving`), the user can notify the solver that a variable, that might be even new, is "relevant" by declaring it as an *observed variable* via `add_observed_var`. When not in state `Solving`, observed variables can be removed via `remove_observed_var`. Note that all IPASIR-UP calls involve observed variables only.

Additional Useful Functions. We propose two additional functions. First, function `phase` (as already implemented in some solvers) allows to force a particular phase of the specified variable when making a decision on that variable. Second, function `is_decision` can be queried for a given variable to determine if it is currently assigned by a decision.

Listing 1 Functions for Configuration and Management (see Section 2.1).

```
// VALID = UNKNOWN | SATISFIED | UNSATISFIED
//
// require (VALID) -> ensure (VALID)
//
void connect_external_propagator (ExternalPropagator * propagator);

// require (VALID) -> ensure (VALID)
//
void disconnect_external_propagator ();

// require (VALID_OR_SOLVING) /\ CLEAN(var) -> ensure (VALID_OR_SOLVING)
//
void add_observed_var (int var);

// require (VALID) -> ensure (VALID)
//
void remove_observed_var (int var);

// require (VALID_OR_SOLVING) -> ensure (VALID_OR_SOLVING)
//
bool is_decision (int observed_var);

// require (VALID_OR_SOLVING) -> ensure (VALID_OR_SOLVING)
//
void phase (int lit);

// require (VALID_OR_SOLVING) -> ensure (VALID_OR_SOLVING)
//
void unphase (int lit);
```

The complete signature of each of these functions is shown in Listing 1. The comments above the functions indicate the IPASIR state of the SAT solver when the function is allowed to be called (see Figure 1a for their relations). The union of states `Unknown`, `SAT`, and `UNSAT` is referred to as `VALID` states here, while the state `VALID_OR_SOLVING` indicates that the function can be called also while the solver is in the `Solving` state.

2.2 Inspecting CDCL via Notifications

We introduce the following three notification functions to capture the changes to the trail (see Listing 2 for signatures). Note that it is acceptable for a SAT solver to delay these notifications (e.g., to notify on assignments only once BCP finished). However, all notifications must happen at the latest before any of the callback functions in Section 2.3 are called.

`notify_assignment`. This function is called when an observed variable is assigned (either by `BCP` or `Decide` or by `Learning` a unit clause). Its first argument is the literal that is satisfied by the assignment, and its second argument is a Boolean flag to indicate when an assignment is *fixed*. A fixed assignment is persistent and the user must ensure that it is *never* undone (even if backtracking would unassign it or some assumptions of the problem are changed).

`notify_new_decision_level`. This function is called on every decision, even if it does not involve an observed variable. It does not report the actual decision or the current decision level – it only reports that a decision happened and thus, the decision level is increased.

notify_backtrack. This function indicates that the solver backtracked to a lower decision level. Its single argument reports the new decision level. All assignments that are not fixed and were made above this new decision level must be treated as unassigned.

2.3 Influencing CDCL via Callbacks

In Section 2.2, we focused on notifying the user about the changes to the trail of the SAT solver. Based on this information, IPASIR-UP allows the user to influence CDCL in various ways via the following callback functions in each of the five states of the search (see Listing 2 for the function signatures).

Decide. Before the solver makes a decision, the callback `cb_decide` allows the user to enforce a user-specific choice of the selected variable and phase. Note that users can inject decisions only after all assumptions are satisfied.

BCP. During unit propagation, the user can provide additional literals to be propagated through the `cb_propagate` callback. Note that this callback returns only a literal to be propagated. The propagating clause is not required at this point.

Conflict Analysis. If during conflict analysis a previous user propagation (see above) turns out to be relevant (i.e., necessary to derive the learnt clause), the solver asks the user for the corresponding reason clause via `cb_add_reason_clause_lit`, one literal at a time. The motivation for such delayed lazy explanation (see [19, 35]) during conflict analysis is to generate and learn only useful clauses.

Solution Analysis. If the solver determines a full assignment without falsifying any present clauses (i.e., a SAT solution is found), `cb_check_found_model` is called. This function tells the solver if the SAT model is consistent with external user constraints. If not, additional clauses can be added to the problem without restarting the search (see below).

Learning. Whenever the solver has finished `BCP` (right before `Decide`), or in case callback `cb_check_found_model` returned false, users can add new clauses to the problem. Callback `cb_has_external_clause` indicates if a new clause is to be added, which is then added via `cb_add_external_clause_lit`, literal by literal. For proof generation, the solver stores these clauses as irredundant original input clauses. In case the learned clause propagates (resp. is falsified) under the current trail, the solver transitions to `BCP` (resp. `Conflict Analysis`). When no more clauses are to be added, the solver continues the search.

2.4 External Propagation with Inprocessing

Our interface IPASIR-UP enables a more fine-grained way of incremental SAT solving, where new clauses may be added not only between two `solve` calls, but also during solving. Ways to combine inprocessing with incremental clause addition was proposed in [17, 33], but their implementations assumed that many clauses are added all at once. Finding an efficient way to implement [17] when new clauses are added one by one during search (as in IPASIR-UP) is intriguing future work. For now, we assume that observed variables are internally frozen, and whenever a variable is added via `add_observed_var`, it is clean w.r.t. the reconstruction stack. This guarantees that no restore step is necessary when external clauses are added.

Listing 2 A C++ example implementation of functions for inspecting and influencing CDCL.

```
class ExternalPropagator {
public:
  virtual ~ExternalPropagator () { }

  virtual void notify_assignment (int lit, bool is_fixed) {}
  virtual void notify_new_decision_level () {}
  virtual void notify_backtrack (size_t new_level) {}

  virtual int cb_decide () { return 0; }
  virtual int cb_propagate () { return 0; }
  virtual int cb_add_reason_clause_lit (int propagated_lit) {
      return 0;
  }
  virtual bool cb_check_found_model (const std::vector<int> & model) {
      return true;
  }

  virtual bool cb_has_external_clause () { return false; }
  virtual int cb_add_external_clause_lit () { return 0; }
};
```

3 Related Work

The main motivation for incremental reasoning is to allow reusing previously learnt information when a similar problem is solved. IPASIR [5] was introduced as a universal interface for incremental SAT solvers, which enables easy integration into applications to take advantage of incremental reasoning without specializing for a specific SAT solver. IPASIR-UP extends IPASIR for use cases that require more fine-grained interaction between the application and the SAT solver during solving. It not only gives the user more comprehensive access to information about the solver state during solving, but allows to influence, and thus, guide its behavior based on user-level information that is not available to the SAT solver.

For instance, adding clauses via IPASIR forces the SAT solver to restart search, delete assumptions, and discard the trail and the implication graph. On the the other hand, adding clauses via `cb_add_external_clause_lit` in IPASIR-UP allows to *continue* the search while keeping all assumptions and backtracking only when a conflict is encountered.

Our proposed interface captures and standardizes functionality that is required by a range of applications, and is thus partially implemented in some tools. An important use case for interaction with the SAT solver as outlined above is the CDCL($\mathcal{T}$) framework [35] for SMT solvers. State-of-the-art SMT solvers based on this framework (e.g., [6, 13, 14]) currently all implement a custom interaction layer with the SAT solver (see, e.g., the SAT worker interface in [13]), which makes replacing these legacy SAT solvers with a state-of-the-art SAT solver highly non-trivial.

The IntelSAT solver [32] implements efficient clause addition on arbitrary decision levels (using *reimplication* to guarantee that no implications are missed on lower decision levels), but does not support external decisions, lazy propagation explanation, nor notifications.

The state-of-the-art ASP solver clingo [18] provides a generic interface to augment the tool with *theory propagators*. It extends the CDCL loop at four locations, with notifications and the ability to add clauses during the search and upon checking the found model. It does not, however, support lazy propagation explanation (i.e, `cb_propagate` with delayed clause addition) and proof generation. The concept of user propagators has also been introduced in the SMT solver z3 [11], mainly to enable users to implement custom theory support.

4 Empirical Evaluation

To show that our interface is effective and efficient in varied use cases, we extended CaDiCaL [10], a state-of-the-art incremental SAT solver which implements the IPASIR interface, with IPASIR-UP. Our extension required ∼800 lines of C++ code in CaDiCaL, accompanied with another ∼700 lines in its model based tester. We provide an evaluation on two representative use cases: enumerating graphs with certain properties via SAT Modulo Symmetries [27], and integrating CaDiCaL as the main CDCL($\mathcal{T}$) SAT engine in the SMT solver cvc5 [6].

4.1 Experiments with SMS

SAT modulo Symmetries (SMS) [23–27] is a recently introduced SAT-based framework for the exhaustive generation of combinatorial objects such as graphs, hypergraphs, or matroids with a given property while excluding isomorphic copies of the same object (isomorph-free). In contrast to a generate-and-test approach, which quickly becomes infeasible due to the extremely fast-growing number of candidate objects, SMS directly generates isomorph-free objects with the desired property. At its core, SMS runs a CDCL solver on a propositional formula that encodes the desired property using object variables.

For instance, if the object is a graph, the graph property is expressed using variables $e_{u,v}$ for each vertex pair u, v indicating existence of an edge between u and v. Isomorphic copies are avoided by guiding the solver to generate canonical objects, e.g., by requiring the adjacency matrix to be lexicographically minimal. Static SAT encodings of lexicographic minimality require an exponential number of clauses [30]. Hence SMS delegates the *minimality check* to an external algorithm invoked whenever the SAT solver decides on an object variable. SMS can perform the minimality check even when many object variables are undecided. This check tests if a minimal object is consistent with the current partial truth assignment. A symmetry-breaking clause is sent back to the CDCL solver if the check fails.

In previous work, SMS used clingo [18], an ASP solver with support for adding custom propagators. The IPASIR-UP interface enables us to replace clingo in SMS with CaDiCaL. We use `cb_has_external_clause` to indicate if we have a symmetry-breaking clause to add and `cb_propagate` to propagate literals. To exhaustively generate all isomorph-free objects with the given property, we add a clause forbidding each object found so far. We can do this via the standard IPASIR interface or IPASIR-UP using callback `cb_check_found_model`.

In the following, we compare the performance of SMS between CaDiCaL+IPASIR-UP and clingo on two graph generation tasks. The first task is to generate up to isomorphism all graphs for a given number n of vertices without additional restrictions, i.e., the formula describing the graph is empty. The second task is to generate up to isomorphism all non-010-colorable graphs with a minimum degree of at least three not containing a cycle of length 4. A graph is 010-colorable if the vertices can be colored with 0 and 1 such that there is no monochromatic edge with color 0 and no monochromatic triangle with color 1.

These graphs are interesting for topics related to the famous Kochen-Specker Theorem from quantum mechanics [2]. For encoding the non-010-colorability, we follow previous work [29]. In contrast to the first task, the encoding is relatively large, even exponential in the number of vertices, and contains auxiliary non-object variables.

For CaDiCaL, the *default* configuration propagates literals and exhaustively enumerates all graphs using the IPASIR-UP interface when possible. Configuration *enum-IPASIR* propagates literals and adds symmetry-breaking clauses via IPASIR-UP but uses IPASIR for enumeration. Configuration *no-prop* corresponds to *default* without propagating literals but learning the clause immediately. Configuration *no-inpro* corresponds to *default* without

Table 1 Enumerating up to isomorphism: all graphs (top) and all KS candidates (bottom).

			CaDiCaL+IPASIR-UP [s]			**Clingo** [s]	
	#vertices	#graphs	*default*	*enum-IPASIR*	*no-prop*	*red*	*irred*
All graphs	6	156	0.01	0.02	0.01	0.02	0.01
	7	1044	0.09	0.13	0.09	0.10	0.09
	8	12346	0.95	1.59	1.00	1.15	1.07
	9	274668	34.24	64.27	34.31	81.67	94.65
	10	12005168	50815.60	109443.72	57616.47	213959.23	196576.58
	#vertices	#graphs	*default*	*no-inpro*	*no-prop*	*red*	*irred*
KS candidates	16	0	10.58	9.14	13.58	25.07	18.56
	17	1	39.82	31.48	44.58	122.28	87.92
	18	0	190.16	59.37	187.29	872.98	493.17
	19	8	1220.51	1253.96	1341.80	10542.41	3348.14
	20	147	13647.66	16449.50	13493.86	67728.42	82871.65

inprocessing on the non-observed variables. For clingo, we either add the clauses as redundant (configuration *red*), i.e., the symmetry-breaking clauses are part of the clause-deletion policy, or the clauses are irredundant (configuration *irred*). Table 1 summarizes the results given the number of vertices in column #vertices. The number of generated graphs is given in column #graphs. All the here presented experiments ran on a cluster equipped with Intel Xeon E5-2640v4 CPUs at 2.40 GHz.

For enumeration, the new interface gives a speedup over IPASIR (Table 1, top): with IPASIR, the search is started at the root level after a model has been found, while with IPASIR-UP, the current trail is preserved and backtracked. The bottom part of Table 1 shows that the versions using CaDiCaL perform better. Inprocessing improves performance on the larger instances, but with less vertices it is more efficient to be turned off. On other SMS applications, we observed clingo and CaDiCaL performing similarly. However, CaDiCaL with IPASIR-UP shows the potential to solve problems outside the other solver's reach.

4.2 Experiments with SMT

Satisfiability Modulo Theories (SMT) solvers serve as the back-end reasoning engine for a variety of applications (e.g., [1,4,12,20,28,34]). The majority of state-of-the-art SMT solvers are based on the CDCL($\mathcal{T}$) framework [35], which tightly integrates theory solvers with a CDCL SAT solver at its core. The CDCL($\mathcal{T}$) SAT engine is queried to find a satisfying assignment of the propositional abstraction of the input formula, which is then iteratively refined until either the assignment is $\mathcal{T}$-consistent or the SAT engine determines unsat.

The CDCL($\mathcal{T}$) framework requires a tight integration with the SAT solver in a way that allows the theory layer to interact with the SAT solver during search, i.e., in an *online* fashion. This is in contrast to other lazy SMT approaches based on the same abstraction/refinement principle that integrate a SAT solver as a *black box*, e.g., lemmas on demand [8,31]. That is, rather than querying the SAT solver for a full satisfying assignment of the propositional abstraction, the theory layer guides the search of the SAT solver until a $\mathcal{T}$-consistent assignment is found or the formula becomes unsatisfiable.

Further, throughout this process, a backward communication channel allows the SAT solver to notify the theory layer about variable assignments, decisions, and backtracks. The theory layer uses this information to derive conflicts, propagate theory literals, or suggest

Table 2 SMT-LIB benchmarks solved by cvc5 and cvc5-ipasirup with a 300 seconds time limit.

	cvc5		cvc5-ipasirup	
Division	solved	time [s]	solved	time [s]
Arith (6,865)	**6,303**	173,628	6,299	176,278
BitVec (6,045)	**5,552**	153,899	5,529	161,482
Equality (12,159)	5,320	2,062,804	**5,322**	2,061,758
Equality+LinearArith (53,453)	45,902	2,288,230	**45,906**	2,288,352
Equality+MachineArith (6,071)	983	1,533,646	**987**	1,532,782
Equality+NonLinearArith (21,104)	**13,314**	2,419,535	13,053	2,486,588
FPArith (3,965)	3,145	268,628	**3,155**	266,245
QF_Bitvec (42,472)	**40,321**	984,880	40,320	985,946
QF_Datatypes (8,403)	8,077	110,704	**8,168**	82,878
QF_Equality (8,054)	8,044	9,394	**8,047**	7,169
QF_Equality+Bitvec (16,585)	15,817	307,558	**16,015**	234,369
QF_Equality+LinearArith (3,442)	**3,388**	23,041	3,381	23,465
QF_Equality+NonLinearArith (709)	627	27,428	**629**	27,598
QF_FPArith (76,238)	76,054	94,487	**76,081**	76,700
QF_LinearIntArith (16,387)	11,670	1,575,635	**12,004**	1,512,696
QF_LinearRealArith (2,008)	1,721	130,408	**1,766**	113,919
QF_NonLinearIntArith (25,361)	13,037	4,094,712	**13,682**	3,840,933
QF_NonLinearRealArith (12,134)	11,166	333,933	**11,238**	316,728
QF_Strings (69,908)	**69,357**	203,677	69,296	230,918
Total (391,363)	339,798	16,796,234	**340,878**	16,426,813

decision variables based on theory-guided heuristics. If theory propagations are involved in deriving a conflict in the SAT solver, the theory layer must provide explanations for the propagated theory literals. If a partial assignment of the propositional abstraction is $\mathcal{T}$-inconsistent, the theory layer sends a lemma to the SAT solver to refine the abstraction.

cvc5 is a state-of-the-art CDCL($\mathcal{T}$) SMT solver widely used in industry and academic projects [6]. It relies on a highly customized version of MiniSat [16] as its core SAT engine, which was extended to support the production of resolution proofs, pushing and popping of assertion levels, and custom theory-guided decision heuristics. The interaction with cvc5's theory layer is directly implemented in MiniSat by various callbacks.

These customizations make it difficult to replace this version of MiniSat with a state-of-the-art SAT solver and take advantage of improvements in SAT solving. Replacing this customized MiniSat with a SAT solver that implements IPASIR-UP enables us to easily switch it out with any other solver that implements the interface. It further has the additional advantage that interaction with the SAT layer is standardized and clean, i.e., no "hacks" have to be added to the SAT solver that may accidentally impact performance.

We integrated CaDiCaL with the IPASIR-UP extension as main CDCL($\mathcal{T}$) SAT engine while fully utilizing the IPASIR-UP notification and callback interface: `notify_assignment` is used to construct the current partial assignment for the observed theory literals; the incremental solver state of cvc5 is managed via `notify_new_decision_level` and `notify_backtrack`, which are utilized to restore its internal state when backtracking decisions; `cb_propagate` and `cb_add_reason_clause_lit` are used for theory propagations and explanations; `cb_decide` to implement custom decision heuristics; `cb_add_external_clause_lit` for adding lemmas and conflicts; and `cb_check_found_model` to check whether the SAT assignment is $\mathcal{T}$-satisfiable. cvc5 further uses `phase` to set the phase for specific variables, and `is_decision` to query if a specific variable was used to make a decision.

The full integration of CaDiCaL as CDCL($\mathcal{T}$) SAT engine of cvc5 required about 700 lines of C++ code on top of cvc5 1.0.5. In the following, we refer to this version of cvc5 with CaDiCaL as the CDCL($\mathcal{T}$) SAT engine as CVC5-IPASIRUP. Note that proof production is not yet supported in CVC5-IPASIRUP, since this requires an extension of the proof infrastructure of cvc5 to support DRAT proofs (MiniSat was customized to emit resolution proofs).

We evaluate the overall performance of CVC5-IPASIRUP against CVC5 version 1.0.5 on all non-incremental benchmarks of the 2022 release of SMT-LIB [7]. We ran this experiment on a cluster equipped with Intel Xeon E5-2650v4 CPUs and allocated one CPU core, 8GB of RAM and a time limit of 300 seconds for each solver and benchmark pair (unknown answers were treated as timeouts). Table 2 shows the number of solved benchmarks and runtime grouped into the divisions defined in SMT-COMP 2022 [36].

Overall, CVC5-IPASIRUP solves 1080 more benchmarks than CVC5 and improves over CVC5 in 13 out of 19 divisions. On the $336,533$ commonly solved benchmarks, CVC5-IPASIRUP ($947,053$s) is 1.16× faster than CVC5 ($1,096,092$s). For quantifier-free divisions, CVC5-IPASIRUP significantly improves over CVC5 in arithmetic logics (+1091) and in logics that combine bit-vectors with arrays (+198). On quantified divisions, CVC5-IPASIRUP's performance is similar to CVC5 except for the UFNIA logic (in division Equality+NonLinearArith), where CVC5-IPASIRUP solves 251 less benchmarks than CVC5. The overall results of CVC5-IPASIRUP are very encouraging, given the fact that the CVC5 code base is tuned for its custom version of MiniSat. This particularly applies to the quantifiers module in CVC5, explaining the UFNIA performance regression. However, the CVC5-IPASIRUP implementation provides a solid baseline to tune and improve cvc5's internals for the IPASIR-UP interface.

5 Summary and Future Work

In this paper, we proposed an extension of the IPASIR interface of SAT solvers to facilitate interactions with the solver *during* the search. We demonstrated the usage and benefits of such an interface in two representative use cases. However, to enable all functionalities of modern SAT solvers, some restrictions were introduced. For example, to enable inprocessing, external clauses can have only observed (i.e., frozen) variables. Further, in our current implementation proof production is only experimental. In future work it needs to be evaluated and extended to support further features such as distinction of redundant and irredundant external clauses.

We believe that both developers of more complex reasoning tools and end-users of SAT solvers can strongly benefit from a unified interface that provides access and control over the details of CDCL methods during incremental problem solving. Though the proposed IPASIR-UP interface provides a sufficient set of functions to cover a very wide range of applications, there are many possible extensions to consider in the future. For example, users might want to decide when to restart the search or where to backtrack upon a conflict. Sharing more information about the internal search statistics, or variable and clause scores could also be valuable. We hope that further discussions and further use cases of IPASIR-UP, for instance in MaxSAT, knowledge compilation or in QBF reasoning, will make it clear what kind of extensions and refinements would be the most practical.

References

1 Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 1–8. IEEE, 2013. URL: `https://ieeexplore.ieee.org/document/6679385/`.

2 Felix Arends, Joël Ouaknine, and Charles W. Wampler. On searching for small Kochen-Specker vector systems. In Petr Kolman and Jan Kratochvíl, editors, *Graph-Theoretic Concepts in Computer Science – 37th International Workshop, WG 2011, Teplá Monastery, Czech Republic, June 21-24, 2011. Revised Papers*, volume 6986 of *LNCS*, pages 23–34. Springer, 2011. `doi:10.1007/978-3-642-25870-1_4`.

3 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiabiliy. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 929–991. IOS Press, 2021. `doi:10.3233/FAIA201008`.

4 John Backes, Ulises Berrueco, Tyler Bray, Daniel Brim, Byron Cook, Andrew Gacek, Ranjit Jhala, Kasper Søe Luckow, Sean McLaughlin, Madhav Menon, Daniel Peebles, Ujjwal Pugalia, Neha Rungta, Cole Schlesinger, Adam Schodde, Anvesh Tanuku, Carsten Varming, and Deepa Viswanathan. Stratified abstraction of access control policies. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification – 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2020. `doi:10.1007/978-3-030-53288-8_9`.

5 Tomás Balyo, Armin Biere, Markus Iser, and Carsten Sinz. SAT race 2015. *Artif. Intell.*, 241:45–65, 2016. `doi:10.1016/j.artint.2016.08.007`.

6 Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022. `doi:10.1007/978-3-030-99524-9_24`.

7 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB), 2023. URL: `http://smt-lib.org`.

8 Clark W. Barrett, David L. Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2002. `doi:10.1007/3-540-45657-0_18`.

9 Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1267–1329. IOS Press, 2021. `doi:10.3233/FAIA201017`.

10 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

11 Nikolaj S. Bjørner, Clemens Eisenhofer, and Laura Kovács. Satisfiability modulo custom theories in Z3. In Cezara Dragoi, Michael Emmi, and Jingbo Wang, editors, *Verification, Model Checking, and Abstract Interpretation – 24th International Conference, VMCAI 2023, Boston, MA, USA, January 16-17, 2023, Proceedings*, volume 13881 of *Lecture Notes in Computer Science*, pages 91–105. Springer, 2023. `doi:10.1007/978-3-031-24950-1_5`.

12 Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In Richard Draves and Robbert van Renesse, editors, *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings*, pages 209–224. USENIX Association, 2008. URL: `http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf`.

13 Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathsat5 SMT solver. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013. doi:10.1007/978-3-642-36742-7_7.

14 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

15 Jo Devriendt, Bart Bogaerts, Broes De Cat, Marc Denecker, and Christopher Mears. Symmetry propagation: Improved dynamic symmetry breaking in SAT. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, pages 49–56. IEEE Computer Society, 2012. doi:10.1109/ICTAI.2012.16.

16 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

17 Katalin Fazekas, Armin Biere, and Christoph Scholl. Incremental inprocessing in SAT solving. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 136–154. Springer, 2019. doi:10.1007/978-3-030-24258-9_9.

18 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos, editors, *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*, volume 52 of *OASIcs*, pages 2:1–2:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/OASIcs.ICLP.2016.2.

19 Ian P. Gent, Ian Miguel, and Neil C. A. Moore. Lazy explanations for constraint propagators. In Manuel Carro and Ricardo Peña, editors, *Practical Aspects of Declarative Languages, 12th International Symposium, PADL 2010, Madrid, Spain, January 18-19, 2010. Proceedings*, volume 5937 of *Lecture Notes in Computer Science*, pages 217–233. Springer, 2010. doi:10.1007/978-3-642-11503-5_19.

20 Patrice Godefroid, Michael Y. Levin, and David A. Molnar. SAGE: whitebox fuzzing for security testing. *Commun. ACM*, 55(3):40–44, 2012. doi:10.1145/2093548.2093564.

21 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 993–1014. IOS Press, 2021. doi:10.3233/FAIA201009.

22 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi:10.3233/SAT190116.

23 Markus Kirchweger, Tomáš Peitl, and Stefan Szeider. Co-certificate learning with SAT modulo symmetries. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence, IJCAI 2023.* AAAI Press/IJCAI, 2023. To appear.

24 Markus Kirchweger, Tomáš Peitl, and Stefan Szeider. A SAT solver's opinion on the Erdős-Faber-Lovász conjecture. In Meena Mahajan and Friedrich Slivovsky, editors, *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04-08, 2023, Alghero, Italy*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. to appear.

25 Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. A SAT attack on Rota's basis conjecture. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 4:1–4:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.SAT.2022.4`.

26 Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. SAT-based generation of planar graphs. In Meena Mahajan and Friedrich Slivovsky, editors, *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04-08, 2023, Alghero, Italy*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. to appear.

27 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 34:1–34:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.CP.2021.34`.

28 K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning – 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer, 2010. `doi:10.1007/978-3-642-17511-4_20`.

29 Zhengyu Li, Curtis Bright, and Vijay Ganesh. A SAT solver + computer algebra attack on the minimum Kochen–Specker problem. Technical report, School of Computer Science at the University of Windsor, November 2022. URL: `https://cbright.myweb.cs.uwindsor.ca/reports/nmi-ks-preprint.pdf`.

30 Eugene M Luks and Amitabha Roy. The complexity of symmetry-breaking formulas. *Annals of Mathematics and Artificial Intelligence*, 41(1):19–45, 2004.

31 Leonardo De Moura and Harald Rueß. Lemmas on demand for satisfiability solvers. In *The 5th International Symposium on the Theory and Applications of Satisfiability Testing, SAT 2002, Cincinnati, USA, May 15, 2002*, 2002.

32 Alexander Nadel. Introducing intel(r) SAT solver. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.SAT.2022.8`.

33 Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Preprocessing in incremental SAT. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012 – 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 256–269. Springer, 2012. `doi:10.1007/978-3-642-31612-8_20`.

34 Aina Niemetz, Mathias Preiner, Clifford Wolf, and Armin Biere. Btor2 , BtorMC and Boolector 3.0. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification – 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 587–595. Springer, 2018. `doi:10.1007/978-3-319-96145-3_32`.

35 Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(T). *J. ACM*, 53(6):937–977, 2006. `doi:10.1145/1217856.1217859`.

36 The International Satisfiability Modulo Theories Competition (SMT-COMP), 2022. URL: `https://smt-comp.github.io/2022`.

37 Hantao Zhang. Combinatorial designs by SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 819–858. IOS Press, 2021. `doi:10.3233/FAIA201005`.

AllSAT for Combinational Circuits

Dror Fried

Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel

Alexander Nadel

Intel Corporation, Haifa, Israel

Faculty of Data and Decision Sciences, Technion, Haifa, Israel

Yogev Shalmon

Intel Corporation, Haifa, Israel

The Open University of Israel, Ra'anana, Israel

Abstract

Motivated by the need to improve the scalability of Intel's in-house Static Timing Analysis (STA) tool, we consider the problem of enumerating all the solutions of a single-output combinational Boolean circuit, called AllSAT-CT. While AllSAT-CT is immediately reducible to enumerating the solutions of a Boolean formula in Conjunctive Normal Form (AllSAT-CNF), our experiments had shown that such a reduction, followed by applying state-of-the-art AllSAT-CNF tools, does not scale well on neither our industrial AllSAT-CT instances nor generic circuits, both when the user requires the solutions to be disjoint or when they can be non-disjoint. We focused on understanding the reasons for this phenomenon for the well-known iterative *blocking* family of AllSAT-CNF algorithms. We realized that existing blocking AllSAT-CNF algorithms fail to generalize efficiently for AllSAT-CT, since they are restricted to Boolean logic. Consequently, we introduce three dedicated AllSAT-CT algorithms that are ternary-logic-aware: a ternary simulation-based algorithm `TALE`, a dual-rail&MaxSAT-based algorithm `MARS`, and their combination. Specifically, we introduce in `MARS` two novel blocking clause generation approaches for the disjoint and non-disjoint cases. We implemented our algorithms in our new tool `HALL`. We show that `HALL` scales substantially better than any reduction to existing AllSAT-CNF tools on our industrial STA instances as well as on publicly available families of combinational circuits for both the disjoint and the non-disjoint cases.

2012 ACM Subject Classification Mathematics of computing → Solvers

Keywords and phrases AllSAT, SAT, Circuits

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.9

Supplementary Material *Software*: `https://github.com/yogevshalmon/allsat-circuits` archived at `swh:1:dir:38b9a6b1c31218fe227ef1848a74ea0a7f37d86e`

Funding *Yogev Shalmon*: This research was partially supported by The Open University of Israel's Research Fund (grant no. 41662).

Acknowledgements We are grateful to Supratik Chakraborty, Alexander Ivrii, Kuldeep Meel, Roberto Sebastiani, Mate Soos and Moshe Vardi for helpful discussions which played an important role in shaping our research. Also, we thank the anonymous reviewers for their comments and useful suggestions.

1 Introduction

Static Timing Analysis (STA) [42] is a crucial step in circuit design process that validates the timing performance of a circuit by checking all possible paths for timing violations. Given a circuit Δ, Intel's STA flow constructs a single-output combinational circuit Γ over Δ's inputs, such that Γ's output is 1 if and only if the inputs have the potential to trigger a timing violation in Δ. Then, the flow enumerates all the solutions in Γ, where every solution is tested for potential violations in the original circuit Δ[1].

[1] Further details are omitted due to IP considerations.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 9; pp. 9:1–9:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Motivated by a recent necessity to increase the scalability of Intel's STA flow due to increasing size of the input circuit, we study the so-called AllSAT-CT problem, which is a vital component in the flow. In *AllSAT-CT*, given a combinational circuit Γ with a single output, the objective is to enumerate all the possible inputs for Γ, for which Γ's output is 1.

AllSAT-CT is an instance of the *AllSAT* problem, in which the goal is to enumerate the solutions of a given Boolean formula. Another instance of AllSAT that received substantially more attention, is *AllSAT-CNF*, in which the input formula is provided in Conjunctive Normal Form (CNF). AllSAT-CNF has various applications, including software testing [22], data mining [7] and network verification [26]. AllSAT-CT can be immediately reduced to AllSAT-CNF by translating the circuit to CNF using, e.g., Tseitin encoding [45], and solved by AllSAT-CNF solvers. There are three main families of approaches to AllSAT-CNF, all implemented in state-of-the-art AllSAT-CNF tools, called herein `Toda` *tools (solvers)*, one per each family [44]. The first family, called *blocking* [28], applies an incremental SAT solver [9, 34] to find a solution (satisfying assignment) σ, then generalizes σ to a solution σ' (by replacing Boolean values by don't-cares, whenever possible), blocks σ' with the so-called *blocking clause* (which usually contains the negation of all the literals assigned 0 or 1 in σ') and iterates. The set of generalized solutions is the (compact) description of all solutions to Γ, typically in a form of a Disjunctive Normal Form (DNF) formula, in which every generalized solution is a cube. The second family of *nonblocking* solvers [14] modifies the SAT solver to enumerate the solutions explicitly without using blocking clauses. The third *BDD-based* family [17] is based on reasoning with Boolean Decision Diagrams (BDDs) [4]. Intel's STA flow used to routinely solve AllSAT-CT by using a BDD-based AllSAT-CNF solver until it ceased to scale due to excessive input size. We tried to apply the `Toda` tools, but they failed to scale either. Our investigation showed that, at least for the blocking algorithms, and independently of our specific industrial application, the existing AllSAT-CNF blocking approaches do not scale well for AllSAT-CT, since their solution generalization components are inherently inefficient, being restricted by Boolean logic semantics. Specifically, since one cannot explicitly assign don't-cares to the variables in Boolean logic, solution generalization's efficiency is hindered.

This insight has led us to introduce three dedicated AllSAT-CT blocking-based algorithms– `TALE`, `MARS` and `DUTY`–that utilize *ternary logic* [35] instead of Boolean, either for generalization only as in `TALE`, or throughout the entire algorithm as in `MARS` and `DUTY`. Notably, ternary logic-based approaches are applied by the Property Directed Reachability (PDR) algorithm for model checking [8, 40], but only for the generalization stage. Moreover, as detailed in [46], there is a distinction between AllSAT algorithms that return *disjoint* solutions, in which no two generalized solutions can overlap, and AllSAT algorithms that return *non-disjoint* solutions, in which such an overlap is allowed. We explicitly designed `MARS` to return either disjoint or non-disjoint solutions, per user request, while `TALE` and `DUTY` return non-disjoint solutions only.

Our first method `TALE` reduces AllSAT-CT to AllSAT-CNF by using Tseitin encoding [45] and applies a *ternary simulation*-based generalization algorithm, commonly used by PDR implementations [8]. Specifically, `TALE` generalizes a given solution by simulating whether reassigning a variable from a Boolean value to a don't-care value, under the ternary-logic semantic, would propagate through the circuit and still set the output to 1.

Our second approach, called `MARS`, also reduces AllSAT-CT to AllSAT-CNF but is using the dual-rail encoding [5] that allows one to preserve the ternary logic semantics in the CNF formula by explicitly representing don't-care values. Specifically, every variable v in the original circuit is mapped to two Boolean *dual-rail* variables (v^+, v^-) in the resulting

CNF, where assigning both v^+ and v^- to 0 corresponds to assigning the original v to a don't-care. With the dual-rail encoding, we trust the SAT solver to return a generalized solution by applying anytime MaxSAT-inspired heuristics [31, 32] to increase the number of don't-cares assigned to the circuit inputs (that is, the number of 0's assigned to their respective dual-rail variables). As mentioned before, MARS can emit both disjoint and non-disjoint solutions, where, unlike in [46], we achieve this by introducing two different blocking clause generation approaches. Furthermore, while generalization with MaxSAT in an auxiliary dual-rail-encoded CNF instance had been applied in PDR [40], our approach of using a single dual-rail-encoded instance throughout the algorithm, combined with a MaxSAT approximation for generalization and blocking clause generation in dual-rail, is novel.

Finally, since MARS uses approximate anytime MaxSAT heuristics, its generalized solutions usually can still be improved. For that, we come up with our third approach called DUTY that adds TALE on top of the MARS to further generalize the solutions obtained by MARS.

We have implemented our approaches in an open-source tool HALL (**H**aifa **All**SAT). We found that HALL scales substantially better than any reduction to existing AllSAT-CNF tools on our industrial STA instances as well as on various publicly available families of combinational circuits. Specifically, MARS is the best-performing algorithm in disjoint mode, while DUTY and TALE are the most scalable ones for industrial and generic instances, respectively, in non-disjoint mode.

The rest of this paper is organized as follows. Sect. 2 presents preliminaries. We review AllSAT in Sect. 3 and present our new algorithms in Sect. 4. Sect. 5 is dedicated to experimental evaluation. Sect. 6 is about related work, while in Sect. 7 we conclude and discuss future work.

2 Preliminaries

We begin by reviewing relevant notions from Boolean and ternary logics. We start with the standard Boolean logic syntax, which, in our context, is common to both logics. Let V be a set of variables. A *literal* l is either a variable $v \in V$, in which case l is *positive*, or a variable's negation $\neg v$, in which case l is *negative*. A formula over V under the standard Boolean logic syntax is in *Conjunctive Normal Form (CNF)* if it is a conjunction of clauses, where a *clause* is a disjunction of literals. Similarly, a formula over V is in *Disjunctive Normal Form (DNF)* if it is a disjunction of cubes, where a *cube* is a conjunction of literals. We assume naturally that no cube and no clause contains both a variable and its negation.

Assuming that circuits are represented in the standard AIGER format [3], a (combinational Boolean single-output) *circuit* Γ with n inputs and m gates is a tuple $\langle I, G, o\rangle$, where $I = \{c_1, \cdots c_n\}$ are *input elements (inputs)*, $G = \{c_{n+1}, \cdots c_{n+m}\}$ are *gate elements (gates)* and $o = c_{m+n+1}$ is a single *output element (output)*. These elements are labeled by a set of variables denoted by $\Gamma(V) = \{v_1, \ldots, v_{n+m+1}\}$. Every input element c_i is labeled by a variable v_i. Every gate element c_k is labeled by a formula $(v_k \leftrightarrow (l_i \wedge l_j))$, where $i, j < k$ and l_i, l_j are literals of variables v_i and v_j respectively. The output element o is labeled with $(v_{n+m+1} \leftrightarrow l_{n+m})$ where l_{n+m} is v_{m+n} or $\neg v_{m+n}$. For simplicity, we identify the elements with their labels (e.g. identify gate c_i as v_i, gate o with v_{n+m+1}, and so on). See Fig. 1a on page 7 for an example of a circuit. Finally, Tseitin encoding [45] generates a CNF from a given circuit Γ, by translating every gate $v \leftrightarrow l_1 \wedge l_2$ to three clauses $(v \vee \neg l_1 \vee \neg l_2) \wedge (\neg v \vee l_1) \wedge (\neg v \vee l_2)$ and adding the unit clause (o) to assert the output.

2.1 Boolean Logic Semantics

In Boolean logic, an assignment $\sigma : V \mapsto \{0,1\}$ assigns each variable to either 0 or 1. An assignment σ is called *total* if σ is a total function and *partial* otherwise. In Boolean logic, the *cardinality* $|\sigma|$ of an assignment σ is the number of variables assigned under σ. Furthermore, in Boolean logic, we say that an assignment ρ *subsumes* the assignment σ, denoted by $\sigma \subseteq \rho$ if whenever ρ assigns a variable v, it holds that σ assigns v as well and $\rho(v) = \sigma(v)$. We then say that σ *extends* ρ. For example, $\rho \equiv \{x_1 := 1\}$ subsumes $\sigma \equiv \{x_1 := 1, x_2 := 0\}$, whereas σ extends ρ. We denote the set of solutions that extend ρ by $sol(\rho)$. An assignment σ for $V = (v_1, \ldots v_n)$ is said to *satisfy* a formula $F(v_1, \ldots v_n)$ in Boolean-logic syntax, if the value of $F(\sigma(v_1), \ldots, \sigma(v_b))$ is 1 under the standard Boolean logic semantic convention. Specifically if F is in CNF then σ satisfies F if it satisfies at least one literal in every clause of F, and if F is in DNF then σ satisfies F if there is at least one cube in F with all its literals satisfied. A *solution* is a (partial or total) satisfying assignment.

Given an assignment σ, we define a cube D^σ as a conjunction of all positive literals v for which $\sigma(v) = 1$, and all negative literals $\neg v$ for which $\sigma(v) = 0$. Same, given a cube D, we define a (possibly partial) assignment σ^D in which $\sigma^D(v) = 1$ for every positive literal v in D and $\sigma^D(v) = 0$ for every negative literal $\neg v$ in D. We then say that σ *induces* D^σ and that D *induces* σ^D. Naturally, σ satisfies D^σ and σ^D satisfies D. We say that two formulas over the same variables, and with the same sets of solutions are *logically equivalent.*

We next define satisfying assignments for circuits. Given a circuit $\Gamma = \langle I, G, o\rangle$, and an assignment σ for $\Gamma(V)$, we say that σ satisfies a circuit element if it satisfies its label. Specifically, σ satisfies a gate $v_k \leftrightarrow l_i \wedge l_j$, if it satisfies the formula $(v_k \leftrightarrow l_i \wedge l_j)$. We say that σ satisfies the circuit Γ if σ satisfies all Γ's gates and $\sigma(o) = 1$.

One can easily show that, given a circuit $\Gamma = \langle I, G, o\rangle$ and a partial assignment $\sigma : I \mapsto \{0,1\}$ to *all* the inputs of Γ, σ can be uniquely extended to a total assignment $\tau_\sigma : I \cup G$ that satisfies all the gates in G. Since τ_σ is fully determined by σ, we can define a solution for partial assignments over *all* the inputs only. Specifically, given a circuit $\Gamma = \langle I, G, o\rangle$ and a partial assignment to its inputs $\sigma : I \mapsto \{0,1\}$, σ is a *solution* if and only if $\tau_\sigma(o) = 1$. Thus, for solving AllSAT-CT, it is sufficient to enumerate solutions defined only over the inputs. This observation holds also for ternary logic, presented next. We say that a circuit and a formula over the circuit's input variables are *logically equivalent* if their sets of solutions is the same. Finally, $\sigma \models T$ denotes that an assignment σ satisfies a formula or a circuit T.

2.2 Ternary Logic Semantics

Ternary logic [35] extends the semantics of Boolean logic with an additional value called *don't-care*, which we denote by X. Formally, in ternary logic, an assignment $\sigma : V \mapsto \{0,1,X\}$ assigns each variable to one of the ternary values $\{0,1,X\}$. To define the value of a formula F in Boolean-logic syntax under a ternary logic assignment σ, we use the ternary logic rules that extend the Boolean logic rules in which $(\neg X = X)$, $(X \wedge 1 = X)$, $(X \wedge 0 = 0)$ and $(X \wedge X = X)$ (we still have $\neg 0 = 1$, $1 \wedge 0 = 0$ and so on, as in the Boolean case). We then say that σ *satisfies* F if σ evaluates F to be 1.

We assume that all the assignments in ternary logic are total. Given a ternary assignment σ, let the *support* of σ, denoted $sup(\sigma)$ to be the set of variables v for which either $\sigma(v) = 0$ or $\sigma(v) = 1$. The *cardinality* of σ is then the size of the support of σ. We say that assignment ρ *subsumes* the assignment σ, denoted by $\sigma \subseteq \rho$, if $\sigma(v) = \rho(v)$ for every $v \in sup(\rho)$. We then say that σ *extends* ρ. For example, $\rho \equiv \{x_1 := 1, x_2 := X\}$ subsumes $\sigma \equiv \{x_1 := 1, x_2 := 0\}$, whereas σ extends ρ. Other definitions, such as logical equivalence, are identical to Boolean logic. The same goes for definitions of circuit assignments with the notable exception of how a gate can be satisfied: consider a circuit $\Gamma = \langle I, G, o \rangle$, an assignment σ and a gate v_k labeled $(v_k \leftrightarrow l_i \wedge l_j)$. Then the gate v_k is satisfied, if it either holds that:

1. $\sigma(v_k) = 1$, where $\sigma(l_i) = 1$ and $\sigma(l_j) = 1$, or
2. $\sigma(v_k) = 0$, where $\sigma(l_i) = 0$ or $\sigma(l_j) = 0$, or
3. $\sigma(v_k) = X$, where $(\sigma(l_i) = X$ and $\sigma(l_j) \neq 0)$ or $(\sigma(l_j) = X$ and $\sigma(l_i) \neq 0)$.

Note that a gate can be assigned to a don't-care value. As before, σ satisfies Γ if σ satisfies all Γ's gates and $\sigma(o) = 1$, and $\sigma \models T$ denotes that σ satisfies a CNF or a circuit T.

2.3 Solution Generalization

Solution generalization is a pivotal notion in our context. Given a solution $\sigma \models T$, where T can be a circuit or a CNF formula, any solution $\sigma' \models T$ which subsumes σ is called a *generalization*, or a *generalized solution* of σ. Since σ' can be extended not only to σ, then by generalizing σ we obtain a compact description σ' of more solutions. To explore less solutions and store them compactly, we are interested in generalizations of small cardinality.

A key observation for our context is that in Boolean logic, generalization can be carried out only by *unassigning* variables, whereas in ternary logic, generalization can be done by *reassigning* variables to X. (This is reflected in our formal definition above, since we defined subsumption differently for the two logics). The above-mentioned difference makes ternary-logic-aware generalization substantially more efficient. Indeed, as we show in Sect. 4, one can easily construct a circuit Γ and a solution σ, such that there exists a generalization of σ in ternary logic that is strictly smaller that any possible generalization in Boolean logic.

3 The AllSAT Problem

AllSAT is the problem of enumerating all the solutions for a given Boolean formula. Designing efficient AllSAT algorithms is a challenge. First, finding even a single solution is already NP-complete, hence an efficient SAT oracle is typically required. Second, since the number of possible solutions can be very large (exponential in the number of the formula's variables), a compact description of the solutions is required.

In this paper, we consider two AllSAT flavours: AllSAT-CNF and AllSAT-CT, depending on the input formula type. In AllSAT-CNF, we are given a CNF formula T, and in AllSAT-CT we are given a combinational circuit T. In both cases the solver is expected to return an enumeration of all solutions for T in a form of a DNF Q that is logically equivalent to the original T. To see why Q is indeed such an enumeration, note that since every cube D in Q induces a satisfying partial assignment σ^D, then every solution that extends σ^D also satisfies D, and therefore satisfies Q. Thus, D compactly describes the set of solutions $sol(\sigma^D)$. Then, as every solution to T satisfies at least a single cube D in Q, we have that Q serves as a compact enumeration of exactly all the solutions for T.

Out of the three main families of AllSAT approaches (blocking [28], non-blocking [14] and BDD-based [17]), mentioned in Sect. 1, we focus in this paper on the iterative *blocking* algorithm [28]. Given a CNF or a circuit, the blocking algorithm uses an incremental SAT solver [9, 34] to find solutions iteratively, where every solution is generalized, and then blocked from subsequently reappearing. This blocking is done by using a so-called *blocking clause* that prevents this solution and perhaps other solutions found so far, from being re-discovered by the SAT solver in subsequent iterations. Formally, given a CNF F and a solution $\sigma \models F$, a clause B is *blocking* if and only if $\sigma \not\models B$ and, for any solution $\tau \models F$ such that $\tau \not\subseteq \sigma$, we have $\tau \models B$ (the latter part is required to ensure correctness, that is, in order not to block solutions, not yet reported to the user). By producing generalized solutions, and adding blocking clauses to the CNF formula, the solver gradually generates all the solutions. The process terminates when the SAT solver returns $UNSAT$, which means that no further solutions can be found, indicating that all possible solutions have been enumerated.

A generic *blocking* algorithm framework for both AllSAT-CNF and AllSAT-CT, adopted from [46], is depicted in Alg. 1. Our AllSAT-CT algorithms, described in Sect. 4, follow this framework. Alg. 1 begins by encoding the circuit Γ into a CNF formula F (if required). Then, following some initialization, the algorithm runs in a loop until the current formula F_i is unsatisfiable (line 5), where, for every iteration i, F_i corresponds to the original formula F, updated with the conjunction of all the blocking clauses generated so far. Inside the loop, the algorithm first finds a solution σ_i for F_i by invoking a SAT solver (line 6). Next, it computes a generalized solution σ'_i by using the GeneralizeSol procedure on σ_i (line 7). Then, the algorithm computes the blocking clause b_i by using the ComputeBlockingCls procedure (line 8). Typically, the blocking clause is the negation of the cube $D^{\sigma'_i}$, induced by the current generalized solution σ'_i (this, however is not always the case; see Sect. 4.2.3). Afterwards, we update the DNF Q with $D^{\sigma'_i}$ and construct the next formula F_{i+1} by adding the newly generated blocking clause to F_i (line 9). Once the loop terminates, the algorithm returns the DNF Q. As one can see, many factors may impact the efficiency of a blocking AllSAT solver dramatically, including the choice of the SAT solver, the circuit encoding for AllSAT-CT, as well as solution generalization and blocking clause computation techniques.

Algorithm 1 Blocking AllSAT Algorithm Template.

```
    Input: Circuit $\Gamma$ or CNF $F$
    Output: Q in DNF with exactly the same solutions as $\Gamma$ or $F$
 1: if the input is a circuit (rather than a CNF) then
 2:     $F :=$ EncodeCircuitToCNF($\Gamma$)      ▷ The input circuit $\Gamma$ is converted to CNF $F$
 3: end if
 4: $i := 1; F_1 := F; Q := \emptyset$
 5: while not UNSAT($F_i$) do
 6:     $\sigma_i :=$ SAT($F_i$)                          ▷ Get the next solution $\sigma_i$
 7:     $\sigma'_i :=$ GeneralizeSol($\sigma_i, \Gamma$)       ▷ $\sigma'_i$ generalizes $\sigma_i$; $\Gamma$ provided only if available
 8:     $b_i :=$ ComputeBlockingCls($\sigma'_i$)        ▷ $b_i$ is disjunction of literals (clause)
 9:     $Q := Q \vee D^{\sigma'_i}, F_{i+1} := F_i \wedge b_i$       ▷ $Q$ is updated by the cube, induced by $\sigma'_i$
10:     $i := i + 1$
11: end while
12: return $Q$                                       ▷ $Q$ may be disjoint or non-disjoint
```

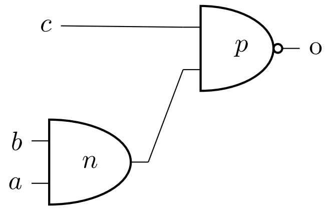
3.1 Disjoint vs. Non-Disjoint Solutions

We next discuss the concepts of disjoint and non-disjoint solutions generation, and their role in the blocking framework. Given a CNF formula or a circuit T and two solutions $\sigma \models T$ and $\tau \models T$, we say that σ and τ are *disjoint*, if there is no solution $\rho \models T$, that extends *both* σ and τ. Otherwise we say that the solutions are *non-disjoint*. Let the DNF Q be a T's AllSAT solution. Two cubes $D_i, D_j \in Q$ are *disjoint* if and only if σ^{D_i} and σ^{D_j} are disjoint. The DNF Q is *disjoint* if all its cubes are pairwise disjoint; otherwise it is *non-disjoint*. One can request an AllSAT solver to generate disjoint or non-disjoint DNFs, where both variants can be of interest, depending on the application [46]. Our industrial application STA does not require the DNF to be disjoint.

By default, the blocking framework Alg. 1 does not guarantee that the resulting DNF Q is disjoint. To produce a disjoint DNF, one can use the following observation from [46]. Let F be the CNF either given as input to Alg. 1 or encoded from a given circuit. We say that a solution τ to F is *blocking-satisfying* if τ also satisfies F_i (comprising F and all the blocking clauses obtained so far). Note that by construction, the solution σ_i, returned by the SAT solver at line 6, is blocking-satisfying, but that does not necessarily mean that the generalized solution σ' is blocking-satisfying as well. It was observed in [46] that, assuming the blocking clauses are constructed as $B := \neg D^{\sigma'_i}$, if every generalized solution σ'_i obtained in line 7 is also blocking-satisfying, then Alg. 1 is guaranteed to return a disjoint DNF. This observation is applicable to TALE, in which generalization does *not* necessarily produce blocking-satisfying solutions, and therefore TALE is not a disjoint solution algorithm. It is not applicable, however, to MARS, since MARS constructs blocking clauses differently; see Sect. 4.2.3 for more details.

4 Ternary Logic-based Algorithms for AllSAT-CT

Towards constructing algorithms that are designated for AllSAT-CT, we first observe that generalizing an existing solution under the Boolean semantics, may result in a solution with larger cardinality (thus less efficient), than when using ternary logic semantics. This is because, in Boolean logic, one cannot explicitly assign don't-care values to the circuit variables. This observation still holds when the circuit is encoded to CNF by using the Tseitin encoding or other encodings under the standard definition that maintains only the Boolean logic semantics (see, e.g. [23]). Thus, using Boolean logic semantics for encoding, may result in missing smaller generalized solutions. We support this observation by the following example.



$$C_1 = (\neg p),$$
$$C_2 = (p \vee \neg n \vee \neg c), C_3 = (\neg p \vee n), C_4 = (\neg p \vee c),$$
$$C_5 = (n \vee \neg a \vee \neg b), C_6 = (\neg n \vee a), C_7 = (\neg n \vee b)$$

(a) $\Gamma = \langle I = \{a, b, c\}$, $G = \{n \leftrightarrow a \wedge b, p \leftrightarrow n \wedge c\}, o \equiv \neg p\rangle$.

(b) Γ's Tseitin encoding into CNF $F = C_1 \wedge \ldots \wedge C_7$.

Figure 1 Illustration for Example 1.

▶ **Example 1.** Consider the circuit Γ in Fig. 1a and the solution $\sigma \equiv \{a := 1, b := 1, c := 0\}$ which satisfies Γ (recall that solutions for circuits are defined over the inputs only). Intuitively, as long as c is assigned 0, the circuit is satisfied, independently from a's and b's values. In ternary logic, σ can be generalized to a solution $\sigma'_t \equiv \{a := X, b := X, c := 0\}$ of cardinality 1 that still satisfies Γ. The corresponding generalized solution in Boolean logic could have been $\tau \equiv \{c := 0\}$. However, note that τ is *not* a solution to Γ, since the gate n cannot remain unassigned. Therefore, in Boolean logic, σ cannot be generalized to a smaller assignment, hence the generalized solution is simply $\sigma'_b = \sigma$ of cardinality 3. To emphasize this point, consider the translation of Γ to a CNF F by Tseitin encoding–in Fig. 1b. One can easily see that unassigning one of a, b or c would render F non-satisfied, since either C_6, C_7 or C_2, would cease to be satisfied, hence generalizing σ in F to a smaller solution is impossible.

To overcome the problem of inefficient generalization in Boolean logic, we propose several blocking AllSAT-CT algorithms that are ternary-logic-aware, as we present next.

4.1 TALE: the Ternary Simulation-based Algorithm

Our first algorithm TALE instantiates the blocking AllSAT algorithm Alg. 1, while borrowing a ternary simulation-based generalization procedure from [8], where it is applied at the generalization stage of the PDR model checking algorithm.

In *ternary simulation* [6, 18], a circuit $\Gamma = \langle I, G, o\rangle$ and an input assignment $\sigma : I \mapsto \{0, 1, X\}$ (which may not satisfy Γ) are given as input. The values of the gates ($v_k \leftrightarrow l_{i<k} \wedge l_{j<k}$) in G, are then computed by iteratively evaluating $l_i \wedge l_j$ based on the ternary semantics outlined in Sect. 2.2. Eventually, ternary simulation returns the evaluation of the circuit, which is the value of o, which can be either $0, 1$ or X.

Given a circuit Γ, the algorithm TALE instantiates Alg. 1 as follows. It first implements EncodeCircuitToCNF by converting Γ into a CNF F using Tseitin encoding. Then, TALE uses the standard incremental SAT-based blocking algorithm to iteratively enumerate and block solutions. The blocking clause computation procedure ComputeBlockingCls simply returns the negation of the cube $D^{\sigma'_i}$, induced by the generalized solution σ'_i. Our focus is on the solution generalization step procedure GeneralizeSol in TALE, which is carried out based on ternary simulation in the original circuit.

Algorithm 2 TALE: GeneralizeSol.

```
   Input: current solution $\sigma_i$ (Boolean or ternary); the original circuit $\Gamma = \langle I, G, o\rangle$
   Output: generalized solution $\sigma'_i$ (in ternary logic) over the circuit inputs
1: $v \in I$: if $v$ is assigned under $\sigma$ then $\sigma'_i(v) := \sigma_i(v)$ else $\sigma'_i(v) := X$    ▷ Initialize $\sigma'_i$
2: for $v \in I; \sigma'_i(v) \neq X$ do                                        ▷ $v$ is a circuit input
3:     $\sigma'_i(v) := X$
4:     Simulate $\sigma'_i$ with ternary-simulation on $\Gamma$ to get an evaluation $eval(\Gamma)$
5:     if $eval(\Gamma) = X$ then
6:         $\sigma'_i(v) := \sigma_i(v)$        ▷ Ternary simulation failed, return the original value of $v$
7:     end if
8: end for
9: return $\sigma'_i$
```

In detail, GeneralizeSol receives the current solution σ_i (σ_i is, originally, in Boolean logic, but Alg. 2 interprets it in ternary logic) and the original circuit Γ and returns a solution $\sigma'_i \models \Gamma$, in ternary logic, that generalizes σ_i. First, Alg. 2 initializes σ'_i from σ_i by setting every variable unassigned in σ_i, to X (line 1). Then, the algorithm iterates through all Γ's inputs (line 2), and, for every input v where $\sigma'_i(v)$ is not X, it tentatively assigns X to v in σ'_i (line 3). Alg. 2 then simulates the updated σ'_i on the circuit C by using ternary simulation (line 4). If the simulation renders the output X, then v cannot be converted to a don't-care, thus the algorithm restores the original value of v, that is $\sigma'_i(v)$ (lines 5–6). Otherwise, v remains X in σ'_i. In the end, the algorithm returns the generalized solution σ'_i (line 9).

TALE does not guarantee that the solutions are disjoint, since while generalization guarantees that the resulting solution σ' satisfies the original circuit (thus, the original CNF), it does not guarantee that the blocking clauses in F_i are satisfied (being unaware of them). Hence, σ' is not necessarily blocking-satisfying (recall Sect. 3.1).

4.2 MARS: the Dual-Rail&MaxSAT-based Algorithm

In this section, we introduce our second algorithm, called MARS. Similarly to TALE, MARS fits into the framework of the blocking algorithm Alg. 1. However, instead of applying a dedicated GeneralizeSol procedure to generalize solutions, it relies on the SAT solver to return an already reasonably generalized solution (hence, MARS implements GeneralizeSol by simply returning $\sigma' \equiv \sigma$). This is achieved by using the so-called dual-rail encoding [5, 21] to convert the circuit to CNF and applying anytime MaxSAT-inspired heuristics [31, 32] in the underlying SAT solver to heuristically generalize solutions. Here, we introduce two new blocking clause generation algorithms for the dual-rail encoding, designed to have MARS return disjoint or non-disjoint solutions, respectively.

Applying the combination of dual-rail encoding and full-blown MaxSAT solving for generalization was proposed in [40], while [38, 10] had previously introduced a closely related method of applying, to the same end, a single SAT invocation that assigns all the decision variables to 0. The latter approach had been proposed in the context of abstraction refinement [38] and minimal model generation for SMT [10], but was also evaluated in the context of PDR in [40]. However, in these works, the main flow creates a separate dual-rail encoded SAT instance for generalization only, while any blocking clauses are created using the standard Tseitin encoding-based technique and added to a Tseitin-encoded CNF instance, maintained by the main flow. Thus, our approach of using a single dual-rail-encoded instance throughout the whole flow, combined with a MaxSAT approximation for generalization and blocking clause generation native to dual-rail encoding, is novel (in addition, our application and the high-level algorithm are completely different from those in [40, 38, 10]). Moreover, our approach of making an AllSAT algorithm return non-disjoint or disjoint solutions, based on different blocking clause generation schemes, is also new.

In what follows, we first describe the dual-rail encoding, followed by the use of a MaxSAT approximation in the generalization process and our blocking clause generation algorithms.

4.2.1 The Dual-Rail Encoding

In dual-rail encoding [5], every variable v in the set of variables $\Gamma(V)$ in a given circuit $\Gamma = \langle I, G, o \rangle$, is mapped to two Boolean *dual-rail* variables (v^+, v^-). This results in a set of variables U that we use to encode the circuit Γ. The dual-rail encoding induces the following one-to-one mapping between a Boolean assignment σ over the dual-rail variables U and a ternary assignment σ over circuit's original variables V (slightly abusing the notation, we reuse σ for both assignments):

1. $\sigma(v) = 1 \iff (\sigma(v^+) = 1$ and $\sigma(v^-) = 0)$
2. $\sigma(v) = 0 \iff (\sigma(v^+) = 0$ and $\sigma(v^-) = 1)$
3. $\sigma(v) = X \iff (\sigma(v^+) = 0$ and $\sigma(v^-) = 0)$
4. the combination $\sigma(v^+) = \sigma(v^-) = 1$ is disallowed.

We now describe the encoding. For a negative literal $l = \neg v$, we use the following notation: $l^+ = v^-$ and $l^- = v^+$. Then, to convert the circuit Γ to CNF, the dual-rail encoding generates the following clauses:

1. For every $v \in V$, we generate the clause $(\neg v^+ \vee \neg v^-)$ to block the disallowed combination.
2. To translate every gate $v \equiv l_i \wedge l_j$, we generate the clauses: $(v^+ \vee \neg l_i^+ \vee \neg l_j^+) \wedge (\neg v^+ \vee l_i^+) \wedge (\neg v^+ \vee l_j^+) \wedge (\neg v^- \vee l_i^- \vee l_j^-) \wedge (v^- \vee \neg l_i^-) \wedge (v^- \vee \neg l_j^-)$.
3. Finally, we generate the unit clause (o^+) to assert the output o.

Note that the dual-rail encoding is heavier than the Tseitin encoding, since it generates twice as many Boolean variables and many more clauses. To summarize, our algorithm MARS implements EncodeCircuitToCNF by using the dual-rail encoding.

4.2.2 MaxSAT Approximation for On-the-Fly Generalization

MaxSAT is a widely used extension of SAT to optimize a linear Pseudo-Boolean function [2]. Given a CNF formula F and a *target bit-vector (target)* $T = \{t_n, t_{n-1}, \ldots, t_1\}$, where each *target bit* t_i is a Boolean variable associated with a strictly positive integer weight w_i, MaxSAT finds a model σ to F that minimizes the following objective function: $\Psi(\sigma) = \sum_{i=1}^{n} \sigma(t_i) \times w_i$. A MaxSAT instance is *unweighted* if and only if all the weights are 1; otherwise it is *weighted*.

It was observed in [40] that, given a solution σ to a circuit $\Gamma = \langle I, G, o\rangle$, one can find a *minimal* generalized solution σ' by the following reduction to unweighted MaxSAT. First, Γ is converted into a CNF formula F using the dual-rail encoding. Next, given $I = \left\{v_1, \ldots, v_{|I|}\right\}$, the vector target T is defined to contain both the dual-rail variables for every input of the original circuit: $T = \left\{v_1^+, v_1^- \ldots, v_{|I|}^+, v_{|I|}^-\right\}$. By using this reduction, invoking a MaxSAT solver over F and T guarantees that the resulting solution, when restricted to the circuit inputs, is of a minimal (but not necessarily *minimum*) cardinality. This is because, by definition, the solver maximizes the number of the Boolean dual-rail variables assigned to 0. Then, since we use clauses to block the assignment $(v^+ := 1, v^- := 1)$ for every pair of the dual-rail variables, we have that every solution σ must assign at least one of the dual-rail variables in every pair to 0. Hence, the minimal solution to F maximizes the number of pairs of the dual-rail variables, in which both variables are assigned to 0, therefore maximizing the number of inputs in the original circuit that are assigned to X.

Since, according to our preliminary experiments, obtaining an optimal MaxSAT solution is computationally expensive, in practice our algorithm `MARS` does not apply full-scale MaxSAT solver to generalize the solutions, but rather uses a standard incremental SAT solver, augmented with two heuristics, applied by anytime MaxSAT solvers [31, 32] to heuristically minimize the solution. Specifically, in the beginning of the search, we boost the score of the (would-be) target variables, which describe the dual-rail variables for the circuit inputs (corresponding to the TSB heuristic in [31, 32]). Additionally, whenever a target variable is chosen by the solver's decision heuristic, we assign it to 0 first (corresponding to the optimistic polarity selection heuristic in [31, 32]). These techniques increase the likelihood of generating heuristically generalized solutions.

4.2.3 Generating Blocking Clauses in `MARS`

Fitting into the blocking algorithm Alg. 1, `MARS` uses the dual-rail encoding to encode the given circuit $\Gamma = \langle I, G, o\rangle$ to a CNF formula F, then trusts the SAT solver, described in Sect. 4.2.2, (invoked at line 6 of Alg. 1) to return an assignment σ for F that already heuristically serves as a generalized solution to Γ (whereas GeneralizeSol, invoked next, simply returns σ). Since the semantics of the dual-rail encoding are ternary-based, we have that σ also assigns every circuit input v to 0, 1 or X, as described in Sect. 4.2.1. We finally explain how `MARS` constructs the blocking clauses, that is, how `MARS` implements ComputeBlockingCls. We also show how with this construction, one can make the distinction between the disjoint and non-disjoint modes, which renders Alg. 1 to return a disjoint or a non-disjoint DNF, respectively.

Towards implementing ComputeBlockingCls, we make use of the observation that it is sufficient for the blocking clause B that we construct to force a change in a value, assigned to at least one of the inputs in $sup(\sigma)$, in order to block the SAT solver from finding σ again. In details, for the disjoint case, B needs to force at least one input $v \in sup(\sigma)$ to *flip* to $\neg\sigma(v)$ (that is, either 0 or 1), where merely changing v to X is not enough. In that way, every future solution ρ, found by the solver, will have at least one input in both $sup(\sigma)$ and

$sup(\rho)$ that has different values. Thus, every solution that extends σ will be different from a solution that extends ρ, as required in the disjoint case. For the non-disjoint case, however, it is sufficient to force at least one input $v \in sup(\sigma)$ to *change* (to either $\neg\sigma(v) \in \{0,1\}$ or X). This will guarantee that every future solution ρ will have at least one input from $sup(\sigma)$ assigned differently than under σ, but altogether σ and ρ can still be non-disjoint; for example, ρ might subsume σ.

We are now ready to present our core algorithms for blocking clause generation for both modes. Observe that for every input v in $sup(\sigma)$, one of the dual-rail variables (v^+, v^-) must be satisfied by σ, while the other one must be falsified (since $(1,1)$ is disallowed).

In the disjoint mode, we set B to be the disjunction of all the dual-rail variables of the form v^ϵ, where $\epsilon \in \{+,-\}$ for which $v \in sup(\sigma)$ and $\sigma(v^\epsilon) = 0$. To see why this works, assume, without loss of generality, that σ assigns some input $v \in sup(\sigma)$ to 1, that is, it assigns (v^+, v^-) to $(1,0)$. Then, B forces every subsequent solution, in which every other input $u \neq v \in sup(\sigma)$ is *not* flipped (that is, u retains the same value as in σ or is assigned X), to assign (v^+, v^-) to $(0,1)$, thus flipping v from 1 to 0.

In the non-disjoint mode, we set B to the disjunction of negative literals of the dual-rail variables of the form $\neg v^\epsilon$, where $\epsilon \in \{+,-\}$ for which $v \in sup(\sigma)$ and $\sigma(v^\epsilon) = 1$. Again to see why this works, assume, without loss of generality, that σ assigns some input v to 1, that is, it assigns (v^+, v^-) to $(1,0)$. Then, B forces every subsequent solution, in which all the other inputs $u \neq v \in sup(\sigma)$ are *unchanged* as compared to σ, to assign (v^+, v^-) to either $(0,1)$ or $(0,0)$, thus changing v by either flipping v to 0 or assigning it X.

Observe that, by construction, any solution τ which is not subsumed by σ is guaranteed to satisfy our blocking clause in both modes. To better understand the difference between the non-disjoint and disjoint modes, consider the following example.

▶ **Example 2.** Consider the circuit Γ in Fig. 1a and the solution $\sigma \equiv \{a := X, b := X, c := 0\}$. Recall that, if $\sigma(c) = 0$, then $\sigma(c^+) = 0$ and $\sigma(c^-) = 1$. In the non-disjoint mode, the generated blocking clause would be $B = (\neg c^-)$. Adding the clause B, allows for the following solution $\tau \equiv \{a := 0, b := 0, c := X\}$, where the solution $\rho \equiv \{a := 0, b := 0, c := 0\}$ is subsumed by both σ and τ, thus result in non-disjoint solutions. On the other hand, in the disjoint mode, the generated blocking clause would be $B = (c^+)$, enforcing c to be assigned 1 in every subsequent solution. Adding the clause B would render $\tau \equiv \{a := 0, b := 0, c := 1\}$, disjoint from σ, the only remaining solution.

4.3 DUTY: Combining MARS and TALE

We finally describe our third algorithm DUTY, which is similar to MARS, except for invoking the ternary simulation-based generalization method of TALE at line 7 in Alg. 1. Such a combination makes sense because of the heuristic nature of the on-the-fly generalization carried out by the SAT solver in MARS. Note that DUTY does not necessarily generate disjoint solutions, since ternary simulation-based generalization does not guarantee that the blocking clauses are satisfied. One may expect DUTY to outperform plain MARS, because ternary simulation over an *almost* minimized solution is expected to be cheap and, hopefully, efficient. It is unclear, however, how DUTY compares to TALE. On one hand, DUTY incurs the overhead of using the dual-rail encoding which generates more clauses and variables than Tseitin encoding. On the other hand, DUTY carries out generalization on-the-fly during SAT solving that leaves substantially less work to ternary simulation.

Another important detail is that, in DUTY, MARS's blocking clause generation is applied in disjoint mode because of the substantially better performance than the non-disjoint mode (inside DUTY) in our preliminary experiments. Informally, sharing too many subsumed solutions slows down the algorithm as it has to enumerate more generalized solutions.

5 Experimental Results

We implemented our algorithms in a new open-source tool `HALL`[2]. Then, we compared these algorithms within `HALL` with the AllSAT-CNF `Toda` tools [44] on our own industrial STA benchmarks (made publicly available), as well as on circuits from the EPFL combinational benchmark suite [1] and random circuits. We carried out the comparison for both the disjoint and non-disjoint cases. In our experiments, we evaluated two criteria: the runtime and the size, where *size* refers to the number of cubes in the resulting DNF.

5.1 Evaluation Setup

`HALL` is written in C++20 on top of `Intel® SAT Solver` [33]. All the experiments were run on machines with 32Gb of memory with Intel® Xeon® processors with 3Ghz CPU frequency. We set the timeout to 3600 seconds.

Since our work considers circuits with one output, we transformed any multi-output circuits into one-output circuit by applying, over all the outputs, either *or* (disjunction) operator (using the utility *aigor* from the *aiger* library [3][3]) or *xor* operator (using our own *aigxor* utility[4]). We omit the conversion time in the results as it is negligible. Furthermore, to evaluate our AllSAT-CT tool against AllSAT-CNF solvers, we translated each circuit from the AIGER format [3] to CNF using the *aigtocnf* utility[5] from the *aiger* library.

5.1.1 Benchmarks

We used the following three benchmark sets:

- **sta_gen** – Static Timing Analysis (STA) industrial set: we generated the following parametrized benchmark family, which encapsulates a variety of real-world STA instances we had encountered. Given the number of inputs N, each formula $F(N)$ consists of a disjunction of subformulas $F_1(N)$ and $F_2(N)$, where each subformula comprises a DNF, conjuncted with the selector v_N or $\neg v_N$. All the cubes in the DNFs have two variables and are pairwise disjoint. The resulting formula looks as follows, where $j = (N-1)/2$:

$$\begin{aligned} F_1(N) &:= ((v_1 \wedge v_2) \vee \ldots \vee (v_{j-1} \wedge v_j)) \wedge v_N \\ F_2(N) &:= ((v_{j+1} \wedge v_{j+2}) \vee \ldots \vee (v_{(N-2} \wedge v_{N-1})) \wedge \neg v_N \\ F(N) &:= F_1(N) \vee F_2(N) \end{aligned}$$

  To satisfy $F(N)$ with the minimal possible solution σ, it is sufficient to satisfy a pair of consecutive variables and assign the selector so as to satisfy its subformula (e.g., assign $\sigma(v_1) = \sigma(v_2) = 1$ and $\sigma(v_N) = 1$). Note that, for $F(N)$ to be well-defined, N must be odd and $N-1$ divisible by 4. While these formulas may be easy to interpret for humans, we note that they are challenging for the automatic solvers that we inspected.
- EPFL combinational benchmark suite [1]: we used the arithmetic and the random_control sets. We created two instances of each set depending on whether the multiple outputs are combined using the operator *or* or the operator *xor*, resulting in the following four sets: **arithmetic_or**, **arithmetic_xor**, **random_control_or**, **random_control_xor**.

[2] `HALL` and all the benchmarks are available at `https://github.com/yogevshalmon/allsat-circuits`.
[3] The library is available at `https://github.com/arminbiere/aiger`.
[4] *aigxor* is available at `https://github.com/yogevshalmon/aiger`.
[5] We used *aigtocnf* with the **–no-pg** option to enforce the translation to preserve the number of solutions.

- Random combinational circuit: we used the *aigfuzz*[6] utility from the AIGER library [3] for generating large combinational circuits. We report results only for the family **large_cir_or**, where the outputs are combined by the operator *or*, since using the operator *xor* resulted in benchmarks which proved to be too difficult for all the solvers.

5.1.2 Solvers

We compared our tool HALL against the three solvers from the Toda repository [44]: BC, NBC and BDD. We tried to get access to the more recent solvers BASolver [47] and AllSATCC [25] but, unfortunately, these tools are not available online, and we could not reach the authors. Below, we list all the solvers and algorithms that we have used, separated by whether they are guaranteed to return only disjoint solutions:

- Disjoint solutions guaranteed:
  - NBC [44]: a non-blocking AllSAT-CNF solver.
  - BDD [44]: a BDD-based AllSAT-CNF solver.
  - MARS: our tool HALL with MARS in disjoint mode.
- Non-Disjoint (that is, disjoint solutions *not* guaranteed):
  - BC [44]: blocking clause-based AllSAT-CNF solver. We used the *SIMPLIFY* and *NONDISJOINT* macros to make the solver return (non-disjoint) partial solutions.
  - TALE: our tool HALL with TALE.
  - MARS: our tool HALL with MARS in non-disjoint mode.
  - DUTY: our tool HALL with DUTY.

5.2 Evaluation of the STA Benchmark Sets

In our first experiment, we used our industrial **sta_gen** benchmark set. We separate our analysis to disjoint and non-disjoint solvers.

5.2.1 Results for Disjoint Solvers

Table 1 Comparing disjoint solvers on the **sta_gen** benchmark family. The first column (N) specifies the number of inputs for each instance. Each pair of columns (T,S) shows the run-time in seconds (where TO stands to a time-out, MO stands for a memory-out and < 1 for a run-time lower than 1 second), and the size of the DNF in the number of cubes.

N	*Disjoint*					
	MARS		NBC		BDD	
	T (sec)	**S**	**T** (sec)	**S**	**T** (sec)	**S**
9	< 1	10	< 1	224	< 1	224
17	< 1	59	< 1	89600	< 1	89600
25	**< 1**	253	2.111	27582464	48.26	27582464
33	**< 1**	1315	570.897	7729971200	MO	–
37	**9.808**	59538	TO	–	MO	–
41	TO	–	TO	–	MO	–

[6] We used the following parameters for aigfuzz: **"-a -c -l -1"**.

The comparison between the disjoint solvers is shown in Table 1. One can see that our `MARS` algorithm is substantially more scalable than the others, but even `MARS` could only handle small formulas with up to 37 inputs. This is because **sta_gen** is a very hard problem in the disjoint mode, since the structure of the formula implies that the number of total solutions is exponential. Consequently, `NBC` and `BDD` that do not iterate over partial solutions, thus return all the total solutions, struggle to scale. `MARS`, which can return partial solutions, scales somewhat better.

5.2.2 Results for Non-Disjoint Solvers

Table 2 shows our results for the non-disjoint solvers comparison. Notably, our tool `HALL`, in all its variants, substantially outperforms `BC`, because of the limitations of generalization in Boolean logic as compared to ternary logic, highlighted in our paper.

Observe also that, for the non-disjoint case, `HALL` was able to solve instances with over 10000 inputs as compared to 37 inputs only for the disjoint case. This is because, for the non-disjoint case (which is the one required in our industrial usage), solving **sta_gen** becomes substantially simpler. Specifically, for every benchmark with N inputs there exists the following DNF solution with $(N-1)/2$ cubes: $Q = (v_1 \wedge v_2 \wedge v_N) \vee \ldots \vee (v_{j-1} \wedge v_j \wedge v_N) \vee \ldots \vee (v_{j+1} \wedge v_{j+2} \wedge \neg v_N) \vee \ldots \vee (v_{N-2} \wedge v_{N-1} \wedge \neg v_N)$.

Table 2 Comparing non-disjoint solvers on **sta_gen**. The first column (N) shows the number of inputs for each instance (inputs from 33 to 2009 are skipped, because of similarity of the results). Each subsequent pair of columns (T,S) shows the solver's run-time in seconds and the size of the resulting DNF in number of cubes.

N	***Non-Disjoint***							
	`TALE`		`MARS`		`DUTY`		`BC`	
	T (sec)	**S**	**T** (sec)	**S**	**T** (sec)	**S**	**T** (sec)	**S**
9	< 1	4	< 1	4	< 1	4	< 1	90
17	< 1	8	< 1	9	< 1	8	< 1	10538
25	< 1	12	< 1	12	< 1	12	866.53	1026771
33	< 1	16	< 1	20	< 1	16	TO	–
2009	< 1	1004	< 1	1154	< 1	1004	TO	–
3009	**< 1**	1504	1.874	1990	**< 1**	1504	TO	–
4009	1.676	2004	2.917	3036	**1.374**	2004	TO	–
5009	3.349	2504	9.83	3629	**2.417**	2504	TO	–
6009	4.403	3004	40.114	20530	**4.181**	3004	TO	–
7009	7.886	3504	10.294	6098	**4.625**	3504	TO	–
9009	28.909	4504	84.817	6889	**11.977**	4504	TO	–
11009	56.877	5504	TO	–	**23.0**	5504	TO	–
13009	96.688	6504	50.361	13704	**29.889**	6504	TO	–

Comparing our algorithms, one can see that `DUTY` outperforms both `TALE` and `MARS`. Notably, `TALE` and `DUTY` return DNFs of the same size, which is the optimal $(N-1)/2$, while `MARS` returns larger DNFs. Apparently, ternary simulation, applied by both `TALE` and `DUTY` (but not by `MARS`), was essential to reduce the size of every cube and also the resulting DNF.

5.3 Evaluation of the EPFL and Random Benchmark Sets

Table 3 summarizes the evaluation results on the EPFL and random families. Specifically, it shows the number of solved instances per each family and solver, where an instance is considered solved, if the solver completed the enumeration all its solutions within the timeout.

Overall, our tool `HALL` solved significantly more instances than the others, where its variant `TALE` is the winner amongst the non-disjoint solver, while `MARS` is the winner amongst the disjoint ones. More specifically, `HALL` is substantially more efficient on instances, where the outputs are joined using the operator *or*. This is because, when the operator *or* is applied over the outputs, the solver can set all the outputs, except for one, to a don't-care. Our dedicated ternary logic-aware algorithms take full take advantage of this property, while other solvers, restricted to Boolean logic, fail to do so. On families, where the outputs are joined using the operator *xor*, the difference is not that significant, where there is one benchmark from the **arithmetic_xor** family, which was solved only by `NBC`.

Table 3 Comparing the number of instances solved from each benchmark set and overall. The first column (Family) shows the benchmark family name. The two subsequent columns provide the number of benchmarks (#Bench) and the average number of inputs for each set (AvgIN). Each of the subsequent columns shows the number of instances solved for the corresponding solver.

Family	**#Bench**	**AvgIN**	***Non-Disjoint***				***Disjoint***		
			`TALE`	`MARS`	`DUTY`	`BC`	`MARS`	`NBC`	`BDD`
arithmetic_or	10	166	**4**	2	3	0	1	1	0
arithmetic_xor	10	166	0	0	0	0	0	**1**	0
random_control_or	10	283	**9**	**9**	**9**	4	7	4	4
random_control_xor	10	283	**5**	**5**	**5**	4	4	4	4
large_cir_or	20	597	**16**	7	**16**	0	7	0	0
Total	60	–	**34**	23	33	8	19	10	8

6 Related Work

AllSAT research has so far been mostly focused on the well studied problem of AllSAT-CNF that is, enumerating all the satisfying assignments of a Boolean formula in CNF [30, 46, 44, 14, 24, 11]; see [44] for an extensive survey. Recent progress in AllSAT-CNF includes [47] that relies on finding backbone variables, and [25] that uses efficient component analysis.

To the best of our knowledge, the only papers that explicitly consider AllSAT-CT are [19, 20, 43]. The first two papers introduce blocking non-disjoint AllSAT-CT algorithms, which utilize a hybrid SAT solver that can carry out conflict analysis and propagation in both CNF formulas and circuits, where the target application is unbounded model checking. The third paper enhances the blocking algorithm with structural analysis. All these algorithms are restricted to Boolean logic in contrast to our ternary logic-aware algorithms. Additionally, the resulting tools are currently unavailable.

A number of solution generalization methods have been proposed in the context of the PDR algorithm for model checking [16, 15, 40, 8, 13]; see [40] for an extensive survey. Finding minimal test cubes in circuit testing is closely related to solution generalization in PDR, where [39, 37] investigate different techniques, including MaxSAT- and MaxQBF-based.

Finally, another closely related problem is one of generating of all the prime implicants, given a Boolean formula, studied in [41, 36, 27]. Informally, in our terminology, an implicant is a cube which implies the original formula (that is, an implicant is a not-necessarily-generalized solution). A prime implicant is a generalized cube, which cannot be generalized further. AllSAT can then be viewed as the problem of finding a (not-necessarily-prime) implicant cover, which is a significantly simpler problem than that of generating all the prime implicants.

7 Conclusion & Future Work

Motivated by the need to improve the scalability of Intel's in-house Static Timing Analysis (STA) tool, we considered the problem of enumerating all the solutions of a single-output combinational Boolean circuit, called AllSAT-CT. We introduced several dedicated ternary logic-based AllSAT-CT algorithms and implemented them in an open-source tool called `HALL`. Our experimental results demonstrated that `HALL` scales substantially better than any reduction to existing AllSAT-CNF tools on our industrial STA instances as well as on various publicly available families of combinational circuits.

For future work, we plan to investigate how to utilize other AllSAT-CNF methods, including the nonblocking technique that modifies the SAT solver to enumerate the solutions explicitly without blocking clauses, for AllSAT-CT. Furthermore, we plan to explore solutions to related problems to utilize the relevant techniques for AllSAT-CT, including dual value propagation in circuits, which is a known method in QBF solving [12] and projected model counting [29]. Specifically, very recently, we became aware of a tool called *dualiza* [29], that, in addition to its main capability of projected model counting, is also capable to enumerate solutions for circuits in the AIGER format. The tool is based on dual calculus. An initial study of ours already suggests promising methods of integrating duality considerations into our algorithms, which we plan to explore.

References

1 Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL combinational benchmark suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.

2 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiabiliy. In *Handbook of Satisfiability – Second Edition*, pages 929–991. IOS Press, 2021.

3 Armin Biere, Keijo Heljanko, and Siert Wieringa. AIGER 1.9 and beyond. Technical Report 11/2, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2011.

4 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

5 Randal E. Bryant, Derek L. Beatty, Karl S. Brace, Kyeongsoon Cho, and Thomas J. Sheffler. COSMOS: A compiled simulator for MOS circuits. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 9–16, 1987.

6 Randall E Bryant. Race detection in mos circuits by ternary simulation, 1983.

7 Imen Ouled Dlala, Saïd Jabbour, Lakhdar Saïs, and Boutheina Ben Yaghlane. A comparative study of SAT-based itemsets mining. In *Research and Development in Intelligent Systems XXXIII – Incorporating Applications and Innovations in Intelligent Systems XXIV. Proceedings of AI-2016.*, pages 37–52, 2016.

8 Niklas Eén, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 125–134, 2011.

9 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT, Proceedings*, 2003.

10 Anders Franzén. *Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT*. PhD thesis, University of Trento, Italy, 2010.

11 Malay K Ganai, Aarti Gupta, and Pranav Ashar. Efficient SAT-based unbounded symbolic model checking using circuit cofactoring. In *IEEE/ACM International Conference on Computer Aided Design,ICCAD.*, pages 510–517, 2004.

12 Alexandra Goultiaeva and Fahiem Bacchus. Exploiting QBF duality on a circuit representation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1791.

13 Alberto Griggio and Marco Roveri. Comparing different variants of the ic3 algorithm for hardware model checking. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 35(6):1026–1039, 2016. doi:10.1109/TCAD.2015.2481869.

14 Orna Grumberg, Assaf Schuster, and Avi Yadgar. Memory efficient all-solutions SAT solver and its application for reachability analysis. In *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD, Proceedings*, 2004.

15 Arie Gurfinkel and Alexander Ivrii. Pushing to the top. In *Formal Methods in Computer-Aided Design, FMCAD*, pages 65–72, 2015.

16 Zyad Hassan, Aaron R Bradley, and Fabio Somenzi. Better generalization in ic3. In *2013 Formal Methods in Computer-Aided Design*, pages 157–164, 2013.

17 Jinbo Huang and Adnan Darwiche. The language of search. *J. Artif. Intell. Res.*, 29:191–219, 2007.

18 James S. Jephson, Robert P. McQuarrie, and Robert E. Vogelsberg. A three-value computer design verification system. *IBM Systems Journal*, 8(3):178–188, 1969.

19 HoonSang Jin, HyoJung Han, and Fabio Somenzi. Efficient conflict analysis for finding all satisfying assignments of a boolean circuit. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS, Proceedings*, 2005. doi:10.1007/978-3-540-31980-1_19.

20 HoonSang Jin and Fabio Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In *Proceedings of the 42nd Design Automation Conference, DAC*, 2005.

21 Daher Kaiss, Marcelo Skaba, Ziyad Hanna, and Zurab Khasidashvili. Industrial strength SAT-based alignability algorithm for hardware equivalence verification. In *Formal Methods in Computer Aided Design (FMCAD)*, pages 20–26, 2007.

22 Sarfraz Khurshid, Darko Marinov, Ilya Shlyakhter, and Daniel Jackson. A case for efficient solution enumeration. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT*, 2003.

23 Petr Kucera and Petr Savický. Backdoor decomposable monotone circuits and their propagation complete encodings. *CoRR*, abs/1811.09435, 2018. arXiv:1811.09435.

24 Bin Li, Michael S. Hsiao, and Shuo Sheng. A novel SAT all-solutions solver for efficient preimage computation. In *2004 Design, Automation and Test in Europe Conference and Exposition*, pages 272–279, 2004.

25 Jiaxin Liang, Feifei Ma, Junping Zhou, and Minghao Yin. Allsatcc: Boosting AllSAT solving with efficient component analysis. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI*, 2022.

26 Nuno P Lopes, Nikolaj Bjørner, Patrice Godefroid, and George Varghese. Network verification in the light of program verification. *MSR, Rep*, 2013.

27 Weilin Luo, Hai Wan, Hongzhen Zhong, Ou Wei, Biqing Fang, and Xiaotong Song. An efficient two-phase method for prime compilation of non-clausal boolean formulae. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021*, pages 1–9. IEEE, 2021. doi:10.1109/ICCAD51958.2021.9643520.

28 Kenneth L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer Aided Verification, 14th International Conference, CAV, Proceedings*, 2002.

29 Sibylle Möhle and Armin Biere. Dualizing projected model counting. In *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI*, pages 702–709, 2018. doi:10.1109/ICTAI.2018.00111.

30 A. Morgado and J. Marques-Silva. Good learning and implicit model enumeration. *Proceedings – International Conference on Tools with Artificial Intelligence, ICTAI*, pages 131–136, 2005.

31 Alexander Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *Formal Methods in Computer Aided Design, FMCAD, Proceedings*, pages 193–202, 2019.

32 Alexander Nadel. Polarity and variable selection heuristics for SAT-based anytime MaxSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2020.

33 Alexander Nadel. Introducing intel(r) SAT solver. In *25th International Conference on Theory and Applications of Satisfiability Testing, SAT*, 2022.

34 Alexander Nadel and Vadim Ryvchin. Efficient SAT solving under assumptions. In *Theory and Applications of Satisfiability Testing – SAT, Proceedings*, 2012.

35 Emil L. Post. Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43, 1921.

36 Alessandro Previti, Alexey Ignatiev, António Morgado, and João Marques-Silva. Prime compilation of non-clausal formulae. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 1980–1988. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/281.

37 Sven Reimer, Matthias Sauer, Paolo Marin, and Bernd Becker. QBF with soft variables. *Electronic Communications of the EASST*, 70, 2014.

38 Jan-Willem Roorda and Koen Claessen. SAT-based assistance in abstraction refinement for symbolic trajectory evaluation. In *Computer Aided Verification, 18th International Conference, CAV, Proceedings*, 2006.

39 Matthias Sauer, Sven Reimer, Ilia Polian, Tobias Schubert, and Bernd Becker. Provably optimal test cube generation using quantified boolean formula solving. In *18th Asia and South Pacific Design Automation Conference, ASP-DAC*, 2013.

40 Tobias Seufert, Felix Winterer, Christoph Scholl, Karsten Scheibler, Tobias Paxian, and Bernd Becker. Everything you always wanted to know about generalization of proof obligations in PDR. under submission. *CoRR*, abs/2105.09169, 2021. arXiv:2105.09169.

41 James R. Slagle, Chin-Liang Chang, and Richard C. T. Lee. A new algorithm for generating prime implicants. *IEEE Trans. Computers*, 1970. doi:10.1109/T-C.1970.222917.

42 Robert B. Hitchcock Sr. Timing verification and the timing analysis program. In *Proceedings of the 19th Design Automation Conference, DAC*, 1982.

43 Abraham Temesgen Tibebu and Görschwin Fey. Augmenting all solution SAT solving for circuits with structural information. In *21st IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS*, pages 117–122. IEEE, 2018. doi:10.1109/DDECS.2018.00028.

44 Takahisa Toda and Takehide Soh. Implementing efficient all solutions SAT solvers. *Journal of Experimental Algorithmics (JEA)*, 2016.

45 Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

46 Yinlei Yu, Pramod Subramanyan, Nestan Tsiskaridze, and Sharad Malik. All-SAT using minimal blocking clauses. *Proceedings of the IEEE International Conference on VLSI Design*, pages 86–91, 2014.

47 Yueling Zhang, Geguang Pu, and Jun Sun. Accelerating All-SAT computation with short blocking clauses. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE*, 2020.

On the Complexity of k-DQBF

Long-Hin Fung ✉
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan

Tony Tan ✉
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan

Abstract

Recently *Dependency Quantified Boolean Formula* (DQBF) has attracted a lot of attention in the SAT community. Intuitively, a DQBF is a natural extension of quantified boolean formula where for each existential variable, one can specify the set of universal variables it depends on. It has been observed that a DQBF with k existential variables – henceforth denoted by k-DQBF – is essentially a k-CNF formula in succinct representation. However, beside this and the fact that the satisfiability problem is NEXP-complete, not much is known about DQBF.

In this paper we take a closer look at k-DQBF and show that a number of well known classical results on k-SAT can indeed be lifted to k-DQBF, which shows a strong resemblance between k-SAT and k-DQBF. More precisely, we show the following.

(a) The satisfiability problem for 2- and 3-DQBF is PSPACE- and NEXP-complete, respectively.
(b) There is a parsimonious polynomial time reduction from arbitrary DQBF to 3-DQBF.
(c) Many polynomial time projections from SAT to languages in NP can be lifted to polynomial time reductions from the satisfiability of DQBF to languages in NEXP.
(d) Languages in the class $\mathsf{NSPACE}[s(n)]$ can be reduced to the satisfiability of 2-DQBF with $O(s(n))$ universal variables.
(e) Languages in the class $\mathsf{NTIME}[t(n)]$ can be reduced to the satisfiability of 3-DQBF with $O(\log\ t(n))$ universal variables.

The first result parallels the well known classical results that 2-SAT and 3-SAT are NL- and NP-complete, respectively.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Dependency quantified boolean formulas, existential variables, complexity

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.10

Funding We acknowledge the generous financial support of Taiwan National Science and Technology Council under grant no. 109-2221-E-002-143-MY3.

Acknowledgements We would like to thank Roland Jiang Jie-Hong for many useful and insightful discussions as well as the anonymous referees for their constructive feedback.

1 Introduction

The last few decades have seen a tremendous development of boolean SAT solvers and their applications in many areas of computing [4]. Motivated by applications in hardware verification and synthesis [19, 3, 29, 17, 5, 7, 22, 16], there have been attempts to build efficient solvers for even higher complexity class such as NEXP. One NEXP-complete logic that recently has attracted a lot of attention is *Dependency Quantified Boolean Formulas* (DQBF). Intuitively, DQBF is a natural extension of Quantified Boolean Formula (QBF) where for each existential variable, one can specify the set of universal variables that it depends on.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 10; pp. 10:1–10:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

It has been observed by various researchers that a DQBF is essentially a succinctly represented boolean formula in conjunctive normal form (CNF), where the information about each clause is encoded inside the matrix of the DQBF and that the number of existential variables in a DQBF corresponds precisely to the width of the clauses of the CNF formula it represents. Such succinctness makes the the satisfiability of DQBF jumps to NEXP-complete [26, 8], compared to "only" NP-complete for SAT. However, beside these facts, not much is known about DQBF and it is natural to ask whether there are more resemblances between DQBF and CNF formulas.

In this paper we show that some well known classical results on SAT can indeed be lifted to DQBF – only with exponential blow-up in complexity due to the succinctness of DQBF. To be more precise, for an integer $k \geqslant 1$, let k-DQBF denote a DQBF with k existentially quantified variables. We establish the following.

(a) The satisfiability of k-DQBF where $k = 1, 2, 3$ is coNP-complete, PSPACE-complete and NEXP-complete, respectively.

(b) There is a parsimonious polynomial time reduction from arbitrary DQBF to 3-DQBF.

Note that (a) parallels the well known classical complexity results on 2-SAT and 3-SAT, i.e., NL-complete and NP-complete and (b) parallels the well known parsimonious polynomial time reduction from an arbitrary boolean formula to a 3-CNF formula.

The fact that DQBF is a succinct representation of CNF formulas actually has the same flavour as the succinct representation of graphs with boolean circuits [14]. In such representation, instead of being given the list of edges in a graph, we are given a boolean circuit $C(\bar{x}, \bar{y})$ where $\bar{x}, \bar{y}$ are vectors of boolean variables with length n. The circuit C represents a graph with $\{0, 1\}^n$ being the set of vertices and two vertices $\bar{u}$ and $\bar{v}$ are adjacent if and only if $C(\bar{u}, \bar{v}) = 1$. It is shown in [25] that many natural NP-complete graph problems become NEXP-complete when succinctly represented. We observe that the proof in [25] can be modified to obtain reductions from DQBF in the following sense.

(c) If there is a projection (in the sense of [33]) from SAT to a graph problem Π, then there is a polynomial time (Karp) reduction from DQBF to the succinctly represented Π.

Briefly, a projection is a special kind of polynomial time reductions first introduced in [33] and it is known that many reductions from SAT to various NP-complete problems are in fact projections [25]. Intuitively, we can view (c) as lifting the reductions from SAT in the class NP to the reductions from DQBF in the class NEXP.

We also observe that DQBF can be used to describe the languages in $\mathsf{NTIME}[t(n)]$ and $\mathsf{NSPACE}[s(n)]$. More precisely, we show the following.

(d) Every language in the class $\mathsf{NTIME}[t(n)]$ can be reduced to 3-DQBF instances with $O(\log t(n))$ universal variables.

(e) Every language in the class $\mathsf{NSPACE}[s(n)]$ can be reduced to 2-DQBF instances with $O(s(n))$ universal variables.

Note that (d) parallels the reductions from the languages in $\mathsf{NTIME}[t(n)]$ to SAT instances with $O(t(n) \log t(n))$ variables and (e) parallels the reductions from the languages in $\mathsf{NSPACE}[s(n)]$ to QBF instances with $O(s(n)^2)$ variables.

Finally, it is open whether there is a natural *bona fide* problem in the class NEXP [25]. In addition to DQBF being a natural extension of QBF and SAT, results (a)–(c) exhibit a strong resemblance between DQBF and CNF formula. Moreover, (d) and (e) show that DQBF can be used to describe both the classes $\mathsf{NTIME}[t(n)]$ and $\mathsf{NSPACE}[s(n)]$, as opposed to the classical results where we need two different logics QBF and SAT to describe them. Combined with the work in [8], we hope they can be convincing evidences for DQBF to be the *bona fide* problem in NEXP, just like SAT in NP and QBF in PSPACE.

Related works. That DQBF is NEXP-complete is proved in [26] and the hardness proof uses an unbounded number of existential variables. It is recently improved in [8] where it is shown that 4 existential variables are sufficient to achieve NEXP-hardness.

Many powerful and interesting techniques have been developed in the last decade for solving DQBF. See, e.g., [2, 12, 15, 23, 39, 36, 38, 21, 31] and the references within. Some recent solvers include iDQ [13], dCAQE [35], HQS [18, 37], DQBDD [32] and Pedant [28, 27]. Recently there is also a DQBF track in the annual SAT competitions [1].

As mentioned earlier, various researchers have observed that a DQBF is essentially a succinctly represented CNF formula, which can be established by *the universal expansion* [6, 13, 3]. Briefly, it removes the universal variables one by one by considering both its values 0 and 1 separately resulting in an exponentially long boolean formula [6]. Based on this insight, some DQBF benchmarks can be constructed by encoding succinctly graph reachability instances and SAT instances [3]. The solver iDQ in [13] is based on universal expansion with additional refinement strategies that remove unnecessary clauses.

A natural extension of QBF to second-order logic that captures the exponential hierarchy is studied in [9, 20]. Naturally a DQBF can be viewed as an existential second-order boolean formula. However, [9, 20] do not study the precise complexity and expressiveness of DQBF itself. In [30] a characterization of a PSPACE subclass of DQBF is introduced. It requires that the dependency sets of all the existential variables are either the same or disjoint and the matrix is in CNF. A close examination shows that it is a Σ_3^p subclass of DQBF, though the precise complexity is still unknown. This subclass is orthogonal to the one in this paper which is based on the number existential variables.

The celebrated Cook-Levin reductions [10, 24] show that every language in $\mathsf{NTIME}[t(n)]$ can be reduced to CNF formulas with $O(t(n)^2)$ variables. The bound on the number of variables was later improved in [11] to $O(t(n)\log t(n))$. The reduction from languages in $\mathsf{NSPACE}[s(n)]$ to QBF with $O(s(n)^2)$ variables is from [34]. In [8] the notion called succinct projection is introduced as a general method to reduce various natural NEXP-complete problems to DQBF. It is the analogue of the Cook-Levin reductions for the class NEXP. Our result (c) shows that the reductions in [25] can be viewed as the "converse" reductions of the ones in [8].

The succinct representation of graphs with boolean circuits was first introduced in [14]. As mentioned earlier, under such representation, many NP-complete problems become NEXP-complete and NL-complete problems become PSPACE-complete [25], using the notion of projections introduced in [33]. To the best of our knowledge, this is the only known technique to lift the results from the class NL and NP to the class PSPACE and NEXP.

Organisation. We first introduce some useful notations and the formal definition of DQBF in Section 2. In Section 3 we study the complexity of DQBF based on the number of existential variables. We discuss how to lift the reductions for NP-complete problems to the class NEXP in Section 4. Then, in Section 5 we show the relations between the class $\mathsf{NTIME}[t(n)]$ and $\mathsf{NSPACE}[s(n)]$ and DQBF. Finally, we conclude with Section 6.

2 Preliminaries

Notations. In this paper we let $\Sigma = \{0,1\}$. We use 0 and 1 to represent the boolean values *false* and *true*, respectively. We will use the symbol a, b, c (possibly indexed) to denote an element in Σ and $\bar{a}, \bar{b}, \bar{c}$ (possibly indexed) to denote a string in Σ^* with $|\bar{a}|$ denoting the length of $\bar{a}$. To avoid clutter, tuples of values from Σ will be written as strings. For example, instead of $(1,0,1,1)$, we will simply write 1011.

We use x, y, z, u, v (possibly indexed) to denote boolean variables and the bar version $\bar{x}, \bar{y}, \bar{z}, \bar{u}, \bar{v}$ (possibly indexed) to denote vectors of boolean variables with $|\bar{x}|$ denoting the length of $\bar{x}$. We implicitly assume that in a vector $\bar{x}$ there is no variable occurring more than once. We write $\bar{z} \subseteq \bar{x}$ when every variable in $\bar{z}$ also occurs in $\bar{x}$.

We write $\varphi(\bar{x})$ to denote a (boolean) formula/circuit with variables/input gates $\bar{x}$. When the variables/input gates are not relevant or clear from the context, we simply write φ.

Let $\varphi(\bar{x})$ be a formula/circuit where $\bar{x} = (x_1, \ldots, x_n)$. Let $\bar{z} = (z_1, \ldots, z_n)$. We write $\varphi[\bar{x}/\bar{z}]$ to denote the formula obtained by substituting each x_i with z_i simultaneously for each $1 \leqslant i \leqslant n$. For a string $\bar{a} = (a_1, \ldots, a_n) \in \Sigma^n$, we write $\varphi[\bar{x} \mapsto \bar{a}]$ to denote the evaluation value of φ when we assign each x_i with a_i.

For $\bar{z} \subseteq \bar{x}$ and $\bar{a} \in \Sigma^{|\bar{x}|}$, we write $\mathsf{prj}_{\bar{x}}(\bar{z}, \bar{a})$ to denote the projection of $\bar{a}$ to the components in $\bar{z}$ according to the order of the variables in $\bar{x}$. For example, if $\bar{x} = (x_1, \ldots, x_5)$ and $\bar{z} = (x_2, x_4, x_5)$, then $\mathsf{prj}_{\bar{x}}(\bar{z}, 00101)$ is 001, i.e., the projection of 00101 to its 2^{nd}, 4^{th} and 5^{th} bits. Note that if $\bar{a} = (a_1, \ldots, a_n)$ and $\bar{x} = (x_1, \ldots, x_n)$, $\mathsf{prj}_{\bar{x}}(x_i, \bar{a})$ is the value a_i.

Dependency quantified boolean formula. A *dependency quantified boolean formula* (DQBF) in prenex normal form is a formula of the form:

$$\Phi := \forall x_1 \cdots \ \forall x_n \ \exists y_1(\bar{z}_1) \cdots \ \exists y_k(\bar{z}_k) \quad \phi \tag{1}$$

where each $\bar{z}_i \subseteq (x_1, \ldots, x_n)$ and ϕ, called *the matrix*, is a quantifier-free boolean formula using variables $x_1, \ldots, x_n, y_1, \ldots, y_k$. The variables $x_1, \ldots, x_n$ are called *the universal variables*, $y_1, \ldots, y_k$ *the existential variables* and each $\bar{z}_i$ *the dependency set* of y_i. We call Φ a k-DQBF, where k is the number of existential variables in Φ. To avoid clutter, sometimes we write Φ as: $\Phi := \forall \bar{x} \ \exists y_1(\bar{z}_1) \cdots \ \exists y_k(\bar{z}_k) \ \phi$, where $\bar{x} = (x_1, \ldots, x_n)$.

A DQBF Φ in the form (1) is *satisfiable*, if there is a tuple $(f_1, \ldots, f_k)$ of functions where $f_i : \Sigma^{|\bar{z}_i|} \to \Sigma$ for every $1 \leqslant i \leqslant k$, and by replacing each y_i with $f_i(\bar{z}_i)$, the formula ϕ becomes a tautology. The tuple $(f_1, \ldots, f_k)$ is called the *satisfying Skolem functions* for Φ. In this case, we also say that Φ is satisfiable by the Skolem functions $(f_1, \ldots, f_k)$.

The problem sat(DQBF) is defined as follows. On input DQBF Φ in the form (1), decide if it is satisfiable. For $k \geqslant 1$, we denote by sat(k-DQBF) the restriction of sat(DQBF) on k-DQBF. As a language, $\mathsf{sat}(\text{DQBF}) := \{\Phi \mid \Phi \text{ is a satisfiable DQBF}\}$ and $\mathsf{sat}(k\text{-DQBF}) := \{\Phi \mid \Phi \text{ is a satisfiable } k\text{-DQBF}\}$.

▶ Remark 1. We may allow the matrix ϕ to be in a *circuit form*, i.e., it is given as a (boolean) circuit with input gates $x_1, \ldots, x_n, y_1, \ldots, y_k$. Such form does not effect the generality of our result since it can be converted to a standard formula form with additional universal variables, but without additional existential variables. See [8, Proposition 1].

▶ Remark 2. A DQBF can be seen a natural generalization of SAT and QBF. Indeed, a boolean formula with variables $y_1, \ldots, y_k$ can be seen as a DQBF without any universal variable and $y_1, \ldots, y_k$ are existential variables with empty dependency set. It is also easy to see that a QBF is just a DQBF where the dependency set form an ordering w.r.t. inclusion, i.e., $\bar{z}_1 \subseteq \bar{z}_2 \subseteq \cdots \subseteq \bar{z}_k$. Indeed, a QBF $\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \forall x_n \exists y_n \ \phi$ can be viewed as a DQBF $\forall x_1 \forall x_2 \cdots \forall x_n \exists y_1(x_1) \exists y_2(x_1, x_2) \cdots \exists y_n(x_1, \ldots, x_n) \ \phi$. Conversely, suppose we have a DQBF Φ as in (1), where $\bar{z}_1 \subseteq \bar{z}_2 \subseteq \cdots \subseteq \bar{z}_k$. Reordering the universal variables, we may assume that $\bar{z}_i = (x_1, \ldots, x_{j_i})$ for each $1 \leqslant i \leqslant k$, where $j_1 \leqslant j_2 \leqslant \cdots \leqslant j_k \leqslant n$. Thus, Φ can be rewritten as the QBF: $\forall x_1 \cdots \forall x_{j_1} \exists y_1 \forall x_{j_1+1} \cdots \forall x_{j_2} \exists y_2 \forall x_{j_2+1} \cdots \forall x_{j_k} \exists y_k \forall x_{j_k+1} \cdots \forall x_n \ \phi$.

Universal expansion. We briefly review the universal expansion method, a useful and well known method for showing that a k-DQBF essentially represents an exponentially large k-CNF formula [6, 13, 3]. Let Φ be k-DQBF as in (1). For each $\bar{a} \in \Sigma^n$, let $\varphi_{\bar{a}}$ be the following boolean formula/circuit.

$$\varphi_{\bar{a}} \ := \ \phi\ [\bar{x}/\bar{a},\ \bar{y}/(f_1(\mathsf{prj}_{\bar{x}}(\bar{z}_1,\bar{a})),\dots,f_k(\mathsf{prj}_{\bar{x}}(\bar{z}_k,\bar{a})))]$$

That is, the variables in $\bar{x}$ in the matrix ϕ are substituted with the values in $\bar{a}$ and each y_i with $f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i,\bar{a}))$. Treating each $f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i,\bar{a}))$ as a boolean variable, $\varphi_{\bar{a}}$ is a standard boolean formula/circuit with k variables and can be rewritten as a k-CNF formula, say, by building its truth table where each row (in the truth table) with 0 value is represented by one clause. By expanding the universal quantifiers, the DQBF Φ can be easily seen to be equivalent to:

$$\bigwedge_{\bar{a}\in\Sigma^n} \varphi_{\bar{a}},$$

where we assume each $\varphi_{\bar{a}}$ is already rewritten in k-CNF.

For our purpose in this paper it is not necessary to convert the entire $\varphi_{\bar{a}}$ into a k-CNF formula. We usually only need to extract one clause at a time from the formula $\varphi_{\bar{a}}$, which is facilitated by the following notation. For each $(\bar{a},\bar{b}) \in \Sigma^n \times \Sigma^k$, where $\bar{a} = (a_1,\dots,a_n)$ and $\bar{b} = (b_1,\dots,b_k)$, we define the clause $C_{\bar{a},\bar{b}}$ as $\ell_1 \vee \cdots \vee \ell_k$, where each literal ℓ_i is as follows.

$$\ell_i \ := \begin{cases} f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i,\bar{a})) & \text{if } b_i = 0 \\ \neg f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i,\bar{a})) & \text{if } b_i = 1 \end{cases}$$

We call $C_{\bar{a},\bar{b}}$ *the clause associated with* $(\bar{a},\bar{b})$ and *the universal expansion of* Φ is defined as:

$$\exp(\Phi) \quad := \bigwedge_{(\bar{a},\bar{b})\in\Sigma^n\times\Sigma^k \text{ s.t. } \phi[(\bar{x},\bar{y})\mapsto(\bar{a},\bar{b})]=0} C_{\bar{a},\bar{b}}$$

Intuitively, $\phi[(\bar{x},\bar{y}) \mapsto (\bar{a},\bar{b})] = 0$ means that we are only interested in the row $\bar{a},\bar{b}$ that yields 0 in the truth table of ϕ. Since the clause $C_{\bar{a},\bar{b}}$ is defined precisely to represent such row 0, it is straightforward to see that $\exp(\Phi)$ is indeed equivalent to $\bigwedge_{\bar{a}\in\Sigma^n} \varphi_{\bar{a}}$, and hence, to Φ.

Alternatively, we can also define the expansion $\exp(\Phi)$ by the following simple rewriting rule. Let $\bar{x} = (x_1,\dots,x_n)$ and $\bar{y} = (y_1,\dots,y_k)$. With additional "fresh" k universal variables $\bar{v} = (v_1,\dots,v_k)$, Φ is equivalent to the following DQBF Φ'.

$$\Phi' \ := \ \forall\bar{x}\ \forall\bar{v}\ \exists y_1(\bar{z}_1)\cdots\ \exists y_k(\bar{z}_k) \qquad \Big(\bigwedge_{i=1}^{k} v_i = y_i\Big) \ \to \ \phi'$$

where ϕ' is $\phi[\bar{y}/\bar{v}]$, i.e., the formula obtained by simultaneously substituting each y_i with v_i in ϕ. Note that ϕ' no longer uses existentially quantified variables. By simple rewriting rule and the expansion on the universal quantifiers, we obtain:

$$\bigwedge_{(\bar{a},\bar{b})\in\Sigma^n\times\Sigma^k} \Big(\bigvee_{i=1}^{k} (\neg b_i \wedge f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i,\bar{a}))) \vee (b_i \wedge \neg f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i,\bar{a})))\Big) \ \vee\ \phi'[\bar{x},\bar{v} \mapsto \bar{a},\bar{b}] \qquad (2)$$

For each $(\bar{a}, \bar{b}) \in \Sigma^n \times \Sigma^k$, when $\phi'[\bar{x}, \bar{v} \mapsto \bar{a}, \bar{b}] = 1$, the conjunct already yields 1. Thus, (2) is equivalent to:

$$\bigwedge_{(\bar{a},\bar{b}) \in \Sigma^n \times \Sigma^k \text{ s.t. } \phi'[\bar{x},\bar{v}\mapsto\bar{a},\bar{b}]=0} \Big(\bigvee_{i=1}^{k} (\neg b_i \wedge f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))) \vee (b_i \wedge \neg f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))) \Big)$$

Since b_i is either 0 or 1, exactly one of $\neg b_i \wedge f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))$ and $b_i \wedge \neg f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))$ evaluates to 0. Thus, it evaluates to either $f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))$ or $\neg f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))$. Therefore, the disjunction $\bigvee_{i=1}^{k} (\neg b_i \wedge f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))) \vee (b_i \wedge \neg f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a})))$ is equivalent to $C_{\bar{a},\bar{b}}$.

3 The complexity of sat(k-DQBF)

In this section we will study the complexity of sat(k-DQBF) for $k = 1, 2, 3$. We will start with the case when $k = 1$ and 2 in Subsection 3.1. The case when $k = 3$ is presented in Subsection 3.2.

3.1 On sat(1-DQBF) and sat(2-DQBF)

We first establish that sat(1-DQBF) is coNP-complete.

▶ **Theorem 3.** *sat(1-DQBF) is coNP-complete.*

Proof. We note that checking whether a boolean formula is tautology is just a special case of sat(1-DQBF) where the existential variable is not used. Since checking tautology is already coNP-hard, the same hardness for sat(1-DQBF) follows immediately. To establish the coNP membership, note that a 1-CNF formula is not satisfiable if and only if it contains two contradicting literals. We will use the same idea for 1-DQBF.

Let Ψ be the following DQBF.

$$\Psi \ := \ \forall \bar{x} \exists y(\bar{z}) \ \psi, \qquad \text{where } \bar{x} = (x_1, \ldots, x_n) \tag{3}$$

It is straightforward to show that there are two contradicting literals in $\exp(\Psi)$ if and only if there is $\bar{a}, \bar{b} \in \Sigma^{n+1}$ such that:

1. $\mathsf{prj}_{(\bar{x},y)}(y, \bar{a}) \ \neq \ \mathsf{prj}_{(\bar{x},y)}(y, \bar{b})$.
2. $\mathsf{prj}_{(\bar{x},y)}(\bar{z}, \bar{a}) \ = \ \mathsf{prj}_{(\bar{x},y)}(\bar{z}, \bar{b})$.
3. $\psi[(\bar{x}, y) \mapsto \bar{a}] = \psi[(\bar{x}, y) \mapsto \bar{b}] = 0$.

The algorithm for accepting unsatisfiable 1-DQBF works as follows. On input Ψ as in (3), guess two assignments $\bar{a}, \bar{b}$ and accept if conditions 1–3 hold. ◀

Next we establish that sat(2-DQBF) is PSPACE-complete.

▶ **Theorem 4.** *sat(2-DQBF) is PSPACE-complete.*

Proof. The PSPACE-hardness is established by reduction from succinct 2-colorability, which is known to be PSPACE-complete [25]. The problem *succinct 2-colorability* is defined as: On input circuit C, decide if the graph represented by C is 2-colorable, i.e., there is coloring of the vertices with 3 colors such that no two adjacent vertices have the same color.

Let $C(\bar{u}, \bar{v})$ be the input circuit, where $|\bar{u}| = |\bar{v}| = n$. We may assume that $\{0, 1\}$ is the set of colors and view a coloring on the vertices as a function $f : \{0, 1\}^n \times \{0, 1\}$. Consider the following 2-DQBF.

$$\Psi \ := \ \forall \bar{x}_1 \forall \bar{x}_2 \ \exists y_1(\bar{x}_1) \exists y_2(\bar{x}_2) \quad \big(\bar{x}_1 = \bar{x}_2 \ \rightarrow \ y_1 = y_2\big) \quad \wedge \quad \big(C(\bar{x}_1, \bar{x}_2) \ \rightarrow \ y_1 \neq y_2\big)$$

where $|\bar{x}_1| = |\bar{x}_2| = n$. Intuitively, it states that y_1 and y_2 must represent the same function and that two adjacent vertices have different colors. It is routine to verify that Ψ is satisfiable if and only if the graph represented by the circuit C is 2-colorable.

To establish the PSPACE-membership, we will use the same idea that 2-SAT is in NL. Note that 2-CNF formula can be rewritten as a conjunction of implications:

$$(\ell_{1,1} \rightarrow \ell_{1,2}) \ \wedge \ (\ell_{2,1} \rightarrow \ell_{2,2}) \ \wedge \ \cdots \ \wedge \ (\ell_{n,1} \rightarrow \ell_{n,2})$$

where each $\ell_{i,j}$ is a literal. In turn, it can be viewed as a directed graph where the literals are the vertices and the implications are the edges. It is not satisfiable if and only if there is a cycle in the graph that contains a literal ℓ and its negation. To check the existence of such a cycle, it suffices to use $O(t)$ space, where t is the number of bits required to remember a literal.

We will use a similar idea to establish the PSPACE-membership of sat(2-DQBF). The detail is as follows. Let Ψ be the input 2-DQBF.

$$\Psi \ := \ \forall \bar{x} \ \exists y_1(\bar{z}_1) \exists y_2(\bar{z}_2) \quad \psi \qquad \text{where } \bar{x} = (x_1, \ldots, x_n).$$

The algorithm works as follows.

- Guess $\bar{a} \in \Sigma^{n+2}$ where $\psi[(\bar{x}, y_1, y_2) \mapsto \bar{a}] = 0$ and a variable $f_i(\mathsf{prj}_{\bar{x},y_1,y_2}(\bar{z}_i, \bar{a}))$ in the clause $C_{\bar{a}}$.
- Guess a series of implications from $f_i(\mathsf{prj}_{\bar{x},y_1,y_2}(\bar{z}_i, \bar{a}))$ to its negation in $\exp(\Psi)$ and vice versa.

To guess a series of implications from $f_i(\mathsf{prj}_{\bar{x},y_1,y_2}(\bar{z}_i, \bar{a}))$ to its negation, it suffices to remember only one literal at a time which requires only $O(n)$ bits. This establishes the PSPACE-membership of sat(2-DQBF). ◀

3.2 On sat(k-DQBF) where $k \geqslant 3$

In this subsection we will consider sat(k-DQBF) where $k \geqslant 3$. We will first establish that sat(3-DQBF) is NEXP-complete. Then, we will show how transform arbitrary DQBF to an equisatisfiable 3-DQBF.

▶ **Theorem 5.** *sat(3-DQBF) is NEXP-complete.*

Proof. The membership is straightforward. The proof for hardness is by reduction from *succinct 3-colorability* which is known to be NEXP-complete [25]. The idea is quite similar to the one in [8] which reduces it to sat(4-DQBF). By a more careful book-keeping, we show that 3 existential variables is enough to encode succinct 3-colorability.

Let $C(\bar{u}, \bar{v})$ be a circuit where $|\bar{u}| = |\bar{v}| = n$ and let $G_C = (V_C, E_C)$ be the graph represented by C. The main idea is simple. We assume that $\{01, 10, 11\}$ is the set of colors and represent a 3-coloring of the graph G_C with a function $f : \{0,1\}^n \times \{0,1\}^2 \rightarrow \{0,1\}$ where for every color $\bar{c} \in \{01, 10, 11\}$, for every vertex $\bar{a} \in \{0,1\}^n$, $f(\bar{a}, \bar{c}) = 1$ if and only if vertex $\bar{a}$ is colored with color $\bar{c}$. We will show that we can construct a 3-DQBF which states that “the coloring must be proper.”

The details are as follows. Let $C(\bar{u}, \bar{v})$ be the input circuit (to succinct 3-colorability) where $|\bar{u}| = |\bar{v}| = n$. Consider the following 3-DQBF Ψ.

$$\begin{aligned}\Psi \ := \ & \forall \bar{x}_1 \forall u_1 \forall v_1 \ \forall \bar{x}_2 \forall u_2 \forall v_2 \ \forall \bar{x}_3 \forall u_3 \forall v_3 \\ & \qquad \exists y_1(\bar{x}_1, u_1, v_1) \ \exists y_2(\bar{x}_2, u_2, v_2) \ \exists y_3(\bar{x}_3, u_3, v_3) \qquad \psi\end{aligned}$$

where $|\bar{x}_1| = |\bar{x}_2| = |\bar{x}_3| = n$ and the formula ψ states the following.

- All y_1, y_2, y_3 are the same function. Formally:

$$\bigwedge_{1 \leqslant i,j \leqslant 3} \Big(\bar{x}_i = \bar{x}_j \ \wedge\ (u_i, v_i) = (u_j, v_j)\Big) \ \rightarrow\ y_i = y_j$$

- No vertex is assigned with the color 00. Formally:

$$(u_1, v_1) = 00 \qquad \rightarrow \qquad y_1 = 0$$

- Every vertex is assigned with exactly one color from $\{01, 10, 11\}$. Formally:

$$\left(\bar{x}_1 = \bar{x}_2 = \bar{x}_3 \ \wedge\ \left(\begin{array}{l} (u_1, v_1), (u_2, v_2), (u_3, v_3) \neq 00 \\ \text{and are pairwise different} \end{array}\right)\right) \ \rightarrow\ \left(\begin{array}{l} \text{exactly} \\ \text{one of} \\ y_1, y_2, y_3 \\ \text{has value 1} \end{array}\right)$$

- Adjacent vertices have different colors. Formally:

$$\Big(C(\bar{x}_1, \bar{x}_2) = 1 \ \wedge\ (u_1, v_1) = (u_2, v_2)\Big) \ \rightarrow\ \Big(y_1 = 0 \ \vee\ y_2 = 0\Big)$$

It is routine to verify that G_C is 3-colorable if and only if Ψ is satisfiable. Moreover, Ψ can be constructed in polynomial time. ◀

Next, we present a parsimonious polynomial time transformation from arbitrary DQBF to 3-DQBF. It is the DQBF analogue of the well known transformation from SAT to 3-SAT.

▶ **Theorem 6.** *There is a parsimonious polynomial time (Karp) reduction from* sat*(DQBF) to* sat*(3-DQBF). In other words, there is a polynomial time algorithm such that: On input DQBF Ψ, it outputs a 3-DQBF Φ such that Ψ and Φ have the same number of satisfying Skolem functions.*

Proof. Before we proceed to give the details, we will first explain the intuition. Consider the following k-DQBF Ψ.

$$\Psi := \forall \bar{x}\ \exists y_1(\bar{z}_1) \cdots\ \exists y_k(\bar{z}_k) \quad \psi \qquad\qquad \text{where } \bar{x} = (x_1, \ldots, x_n)$$

Let $n_i = |\bar{z}_i|$, for each $1 \leqslant i \leqslant k$. We will encode the satisfying Skolem functions $(f_1, \ldots, f_k)$ for Ψ as one function $g : \Sigma^n \times \Sigma^k \to \Sigma$ as follows. For every $\bar{a} \in \Sigma^n$, for every $\bar{b} = (b_1, \ldots, b_k) \in \Sigma^k$,

$$g(\bar{a}, \bar{b}) \ = \ 1 \quad \text{if and only if} \quad b_i = f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a})) \qquad\qquad \text{for every } 1 \leqslant i \leqslant k$$

We call such function g *the encoding of* $(f_1, \ldots, f_k)$. Note that for a function $g : \Sigma^n \times \Sigma^k \to \Sigma$ to properly encode k functions $(f_1, \ldots, f_k)$, it has to satisfy the following "functional property":

For every $\bar{a} \in \Sigma^n$, there is exactly one $\bar{b} \in \Sigma^k$ such that $g(\bar{a}, \bar{b}) = 1$.

Unfortunately DQBF by itself is not strong enough to state such property. For this, we need another type of encoding that can be expressed with 3-DQBF.

We first introduce a few terminology and notations. For $\bar{b}, \bar{c} \in \Sigma^k$, we denote by $\bar{b} \leqslant_{\text{lex}} \bar{c}$, if $\bar{b}$ is "lexicographically" less than or equal to $\bar{c}$.[1] Note that one can easily write a (boolean) formula $\varphi(\bar{x}_1, \bar{x}_2)$, where $|\bar{x}_1| = |\bar{x}_2| = k$ such that $\varphi[(\bar{x}_1, \bar{x}_2) \mapsto (\bar{b}, \bar{c})] = 1$ if and only if $\bar{b} \leqslant_{\text{lex}} \bar{c}$. We denote by $\bar{b} - 1$ and $\bar{b} + 1$ the induced predecessor and successor of $\bar{b}$ in the lexicographic ordering of Σ^k (when $\bar{b} \neq 0^k$ and $\bar{b} \neq 1^k$, respectively).

The monotonic encoding of the functions $(f_1, \ldots, f_k)$ is a function $h : \Sigma^n \times \Sigma^k \to \Sigma$ such that for every $\bar{a} \in \Sigma^n$, the following holds.

[1] This is the standard lexicographic ordering on Σ^k where 0 is "less than" 1.

$\bar{x} = (x_1, \dots, x_n)$	$\bar{v} = (v_1, \dots, v_k)$	$h(\bar{x}, \bar{v})$	
$\bar{a}$	0^k	0	all zeroes
$\vdots$	$\vdots$	$\vdots$	
$\bar{a}$	$\bar{b} - 1$	0	
$\bar{a}$	$\bar{b}$	1	all ones
$\bar{a}$	$\bar{b} + 1$	1	
$\vdots$	$\vdots$	$\vdots$	
$\bar{a}$	1^k	1	

Figure 1 Here $\bar{b} \in \Sigma^k$. The notations $\bar{b} - 1$ and $\bar{b} + 1$ denote the predecessor and the successor of $\bar{b}$, respectively, according to the lexicographic ordering of Σ^k. The function $h : \Sigma^n \times \Sigma^k \to \Sigma$ is monotone (w.r.t. $\bar{v}$), i.e., if $h(\bar{a}, \bar{b}) = 1$, then $h(\bar{a}, \bar{c}) = 1$ for every $\bar{c}$ greater than $\bar{b}$ lexicographically. Such h encodes a function $g : \Sigma^n \times \Sigma^k \to \Sigma$ where $g(\bar{a}, \bar{b}) = 1$ if and only if $h(\bar{a}, \bar{b} - 1) = 0$ and $h(\bar{a}, \bar{b}) = 1$.

- For every $\bar{b}, \bar{c} \in \Sigma^k$, $h(\bar{a}, \bar{b}) \leqslant h(\bar{a}, \bar{c})$ whenever $\bar{b} \leqslant_{\text{lex}} \bar{c}$. That is, it is monotonic w.r.t. to the last k bits.
- For every $\bar{b} = (b_1, \dots, b_k) \in \Sigma^k$ such that $b_i = f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))$, for every $1 \leqslant i \leqslant k$, the following holds.
  - $h(\bar{a}, \bar{c}) = 0$, for every $\bar{c} \in \Sigma^k$ where $\bar{c} <_{\text{lex}} \bar{b}$.
  - $h(\bar{a}, \bar{c}) = 1$, for every $\bar{c} \in \Sigma^k$ where $\bar{b} \leqslant_{\text{lex}} \bar{c}$.

Intuitively, if h is the monotonic encoding of $(f_1, \dots, f_k)$, the value $\bar{b} = (b_1, \dots, b_k)$ where $b_i = f_i(\mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{a}))$ for every $1 \leqslant i \leqslant k$ can be identified as the lexicographically smallest $\bar{b}$ such that $g(\bar{a}, \bar{b}) = 1$. See Figure 1 for an illustration.

Note that if $h : \Sigma^n \times \Sigma^k \to \Sigma$ is the monotonic encoding of $(f_1, \dots, f_k)$, we can recover the encoding of $(f_1, \dots, f_k)$. Indeed, define the function $g : \Sigma^n \times \Sigma^k \to \Sigma$ as follows.

- If $h(\bar{a}, 0^n) = 0$, then $g(\bar{a}, \bar{b}) = 1$ if and only if $h(\bar{a}, \bar{b} - 1) = 0$ and $h(\bar{a}, \bar{b}) = 1$.
  In this case, note that there is exactly one $\bar{b}$ such that $h(\bar{a}, \bar{b} - 1) = 0$ and $h(\bar{a}, \bar{b}) = 1$. Thus, there is exactly one $\bar{b}$ such that $g(\bar{a}, \bar{b}) = 1$.
- If $h(\bar{a}, 0^n) = 1$, then $g(\bar{a}, 0^n) = 1$ and for every $\bar{b} \neq 0^n$, $g(\bar{a}, \bar{b}) = 0$.

Since h is the monotonic encoding of $(f_1, \dots, f_k)$, it is immediate that g is the encoding of $(f_1, \dots, f_k)$.

We now give the details of the reduction from $\mathsf{sat}(\text{DQBF})$ to $\mathsf{sat}(\text{3-DQBF})$. On input $\Psi := \forall \bar{x} \exists y_1(\bar{z}_1) \cdots \exists y_k(\bar{z}_k)\ \psi$, where $\bar{x} = (x_1, \dots, x_n)$, it outputs the following DQBF:

$$\Phi \ := \ \forall \bar{x}_1 \forall \bar{v}_1 \ \forall \bar{x}_2 \forall \bar{v}_2 \ \forall \bar{x}_3 \forall \bar{v}_3 \ \exists p_1(\bar{x}_1, \bar{v}_1) \exists p_2(\bar{x}_2, \bar{v}_2) \exists p_3(\bar{x}_3, \bar{v}_3) \quad \phi$$

where $|\bar{x}_i| = n$ and $|\bar{v}_i| = k$ for each $1 \leqslant i \leqslant 3$ and ϕ states the following.

- p_1 and p_2 represent the monotonic encoding of the Skolem functions $(f_1, \dots, f_k)$ for Ψ (if exist).
- p_3 represents the encoding of the Skolem functions $(f_1, \dots, f_k)$ for Ψ (if exists).

The details of ϕ are as follows. Let $\bar{x}_i = (x_{i,1}, \dots, x_{i,n})$ and $\bar{v}_i = (v_{i,1}, \dots, v_{i,k})$ for every $1 \leqslant i \leqslant 3$.

(1) p_1 and p_2 represent the same function. Formally:

$$(\bar{x}_1 = \bar{x}_2 \quad \wedge \quad \bar{v}_1 = \bar{v}_2) \quad \to \quad (p_1 = p_2)$$

(2) The function represented by p_1 (and p_2) is a monotonic w.r.t. the bits in $\bar{v}_1$. Formally:

$$(\bar{x}_1 = \bar{x}_2 \quad \wedge \quad \bar{v}_1 \leqslant_{\text{lex}} \bar{v}_2) \quad \to \quad (p_1 \leqslant p_2)$$

(3) That $\bar{v}_1 = 1^k$ implies $p_1 = 1$. Formally:

$$\bar{v}_1 = 1^k \quad \to \quad p_1 = 1$$

Note that this condition implies that the fact that if p_1 represents a function $h : \Sigma^n \times \Sigma^k \to \Sigma$, then for every $\bar{a} \in \Sigma^n$, there is $\bar{b} \in \Sigma^k$ such that the value $h(\bar{a}, \bar{b}) = 1$. This is because p_1 represents a monotonic function w.r.t. $\bar{v}_1$.

(4) The function represented by p_3 is the encoding of the k functions whose monotonic encoding is represented by p_1 (and p_2). Formally:

$$\begin{aligned}
&\Big((\bar{x}_1 = \bar{x}_2 = \bar{x}_3) \ \wedge \ (\bar{v}_1 + 1 = \bar{v}_2 = \bar{v}_3)\Big) \quad \to \quad \Big((p_1 = 0 \ \wedge \ p_2 = 1) \leftrightarrow p_3 = 1\Big) \\
\wedge\ &\Big((\bar{x}_2 = \bar{x}_3) \ \wedge \ (\bar{v}_2 = \bar{v}_3 = 0^k)\Big) \quad \to \quad \Big(p_2 = 1 \leftrightarrow p_3 = 1\Big)
\end{aligned}$$

(5) The functions $(f_1, \ldots, f_k)$ encoded by p_1, p_2, p_3 respect the dependency set $\bar{z}_i$ for every $1 \leqslant i \leqslant k$. Formally:

$$\begin{aligned}
&\Big((\bar{x}_1 = \bar{x}_2 \ \wedge \ \bar{v}_1 + 1 = \bar{v}_2 \ \wedge \ p_1 = 0 \ \wedge \ p_2 = 1\ ) \ \wedge \ p_3 = 1\Big) \\
&\qquad\qquad \to \bigwedge_{1 \leqslant i \leqslant k} \mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{x}_2) = \mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{x}_3) \ \to \ v_{2,i} = v_{3,i} \\
\wedge\ &\Big((\bar{v}_2 = 0^k \ \wedge \ p_2 = 1) \ \wedge \ p_3 = 1\Big) \\
&\qquad\qquad \to \bigwedge_{1 \leqslant i \leqslant k} \mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{x}_2) = \mathsf{prj}_{\bar{x}}(\bar{z}_i, \bar{x}_3) \ \to \ v_{2,i} = v_{3,i}
\end{aligned}$$

(6) The functions $(f_1, \ldots, f_k)$ encoded by p_3 is indeed a satisfying Skolem functions for Ψ. Formally:

$$p_3 = 1 \quad \to \quad \psi[(\bar{x}, y_1, \ldots, y_k)/(\bar{x}_3, \bar{v}_3)]$$

It is routine to verify that Ψ and Φ are equisatisfiable. Note also that every Skolem functions $(f_1, \ldots, f_k)$ for Ψ is uniquely represented by their encoding and monotonic encoding. Conversely, every encoding and monotonic encoding represented by p_3 and p_1, p_2 uniquely represented the Skolem functions $(f_1, \ldots, f_k)$ for Ψ. Thus, Ψ and Φ have the same number of satisfying Skolem functions. By inspection, the 3-DQBF Φ can be constructed in polynomial time. ◀

4 Lifting the projections in the class NP to the class NEXP

In this section we will establish the relations between the so called *projections* (from SAT to NP-complete graph problem Π) and polynomial time reductions (from sat(DQBF) to the succinctly represented Π). We first recall the definition of projections [33, 25]. Let $\xi : \Sigma^* \to \Sigma^*$ be a reduction from a language L to another language K. We say that ξ is a *projection*, if the following holds.

- There is a polynomial $p(n)$ such that for every $w \in \Sigma^*$, the length of $\xi(w)$ is $p(|w|)$.
- There is a polynomial time (deterministic) algorithm $\mathcal{A}$ such that on input 1^n and $1 \leqslant i \leqslant p(n)$, where i is in the binary representation, the output $\mathcal{A}(1^n, i)$ is one of the following:
  - a value (either 0 or 1);
  - a variable x_j (appropriately encoded) where $1 \leqslant j \leqslant n$;
  - the negation of a variable $\neg x_j$ (appropriately encoded) where $1 \leqslant j \leqslant n$;

  such that if $z_1 \cdots z_{p(n)}$ are the output $\mathcal{A}(1^n, 1), \ldots, \mathcal{A}(1^n, p(n))$, the following holds. For every $w = b_1 \cdots b_n \in \Sigma^n$:

  $$\xi(w) \;=\; z_1 \cdots z_{p(n)}[(x_1, \ldots, x_n) \mapsto (b_1, \ldots, b_n)]$$

  where $z_1 \cdots z_{p(n)}[(x_1, \ldots, x_n) \mapsto (b_1, \ldots, b_n)]$ is the 0-1 string obtained by substituting each x_i with b_i.

The intuitive meaning of the algorithm $\mathcal{A}(1^n, i)$ is as follows. The i^{th} bit of the output of the reduction ξ on an input of length n is either 0 or 1 or the j^{th} bit of the input (when $\mathcal{A}(1^n, i) = x_j$) or the complement of the j^{th} bit of the input (when $\mathcal{A}(1^n, i) = \neg x_j$). Note also that if there is a projection ξ from L to K, then there is a polynomial time reduction from L to K, where on input w, we compute each bit in $\xi(w)$ by computing $\mathcal{A}(1^{|w|}, i)$ for every $1 \leqslant i \leqslant p(|w|)$. Almost all know reductions from SAT to graph problems are, in fact, projections. We recall the following result from [25] and briefly review the proof.

▶ **Theorem 7** ([25]). *If there is a projection from SAT to a graph problem* Π*, then the succinct version of* Π *is NEXP-hard.*

Proof. We assume that a graph $G = (V, E)$ is encoded as 0-1 string of length $|V|^2$ representing the adjacency matrix of G. Let $L \in$ NEXP. Let $\mathcal{M}$ be the NTM that accepts L in time $2^{p(n)}$ for some polynomial $p(n)$. For $w \in \Sigma^*$, let F_w denote the CNF formula obtained by applying the standard Cook-Levin reduction on w (w.r.t. the NTM $\mathcal{M}$). Assuming that F_w is encoded as 0-1 string, the length of $F(w)$ is $2^{q(n)}$ for some polynomial $q(n)$.

We can design a polynomial time deterministic algorithm $\mathcal{A}$ that on input w and two indexes i, j, determine if a literal ℓ_i appears in clause C_j in the formula F_w. Note that i and j can be encoded in binary representation with $p(n)$ bits. We can easily modify $\mathcal{A}$ into another algorithm $\mathcal{A}'$ such that on input w and index $1 \leqslant i \leqslant 2^{q(n)}$, output the bit-$i$ in the formula F_w.

Let ξ be the projection from SAT to a graph problem Π. Suppose for a formula F, the graph $\xi(F)$ has $r(|F|)$ vertices, for some polynomial $r(n)$. Using ξ, we can design a polynomial time algorithm $\mathcal{B}$ that on input w and index $1 \leqslant i, j \leqslant r(2^{q(n)})$ (in binary), output an index i' (in binary) such that the bit-i' in F_w is the same as the (i, j)-entry in the adjacency matrix of $\xi(F_w)$. We can then combine both algorithms $\mathcal{A}'$ and $\mathcal{B}$ to obtain another algorithm $\mathcal{C}$ such that on input w and indexes $1 \leqslant i, j \leqslant r(2^{q(n)})$, it outputs the (i, j)-entry in the adjacency matrix of $\xi(F_w)$. Note that when the length of the input is fixed, we can construct in polynomial time the boolean circuit representing the algorithm $\mathcal{C}$.

Now, the reduction from L to succinct Π works as follows. On input w, it constructs the boolean circuit for $\mathcal{C}$ where the input length is fixed to $|w| + 2 \cdot \log r(2^{q(n)})$ and the first $|w|$ input gates are fed with w. Note that the output circuit represents the graph $\xi(F_w)$. Thus, we obtain the reduction from L to succinct Π. ◀

We show that it can actually be stated as follows.

▶ **Corollary 8.** *If there is a projection from SAT to a graph problem Π, then there is a polynomial time (Karp) reduction from* sat*(DQBF) to the problem Π in succinct representation.*

Proof. We actually just follow the proof in [25] with a slight modification on the definition of $\exp(\Psi)$. Let Ψ be a DQBF with matrix ψ. We modify the definition of the clause $C_{\bar{a},\bar{b}}$ as follows.

- If $\psi[\bar{x}, \bar{v} \mapsto \bar{a}, \bar{b}] = 0$, we set $C_{\bar{a},\bar{b}}$ as in Section 2.
- If $\psi[\bar{x}, \bar{v} \mapsto \bar{a}, \bar{b}] = 1$, we set $C_{\bar{a},\bar{b}}$ as a trivial clause such as $(\neg x \vee x \vee \cdots \vee x)$ for some arbitrary variable x.

Under such definition, we can easily design an algorithm $\mathcal{A}'$ that on input DQBF Ψ and index i, output the i^{th} bit in the formula $\exp(\Psi)$. The reduction from sat(DQBF) to succinct Π can be obtained in exactly the same way as in Theorem 7. ◀

5 The relations between NTIME$[t(n)]$, NSPACE$[s(n)]$ and DQBF

In this section we will show how to reduce the languages in the class NTIME$[t(n)]$ and NSPACE$[s(n)]$ to DQBF. We implicitly assume that the functions $t(n)$ and $s(n)$ are time/space constructible. We start with the following theorem which has been proved in [8].[2]

▶ **Theorem 9** ([8, Theorem 1]). *For every $L \in$ NTIME$[t(n)]$, there is a (deterministic) algorithm $\mathcal{A}$ that runs in time polynomial in n and $\log t(n)$ such that on input word w, it outputs a DQBF Ψ such that $w \in L$ if and only if Ψ is satisfiable. Moreover, the output DQBF Ψ uses $O(\log t(n))$ universal variable and $O(1)$ existential variables, where n is the length of the input w.*

The constant hidden in $O(1)$ in Theorem 9 depends on the number of states, tapes and tape symbols of the Turing machine M that decides L. Combining Theorem 6 and 9, we obtain the following corollary.

▶ **Corollary 10** ([8, Theorem 1]). *For every $T(n) \geqslant n$, for every $L \in$ NTIME$[t(n)]$, there is a (deterministic) algorithm $\mathcal{A}$ that runs in time polynomial in n and $\log t(n)$ such that on input word w, it outputs a 3-DQBF Ψ such that $w \in L$ if and only if Ψ is satisfiable. Moreover, the output DQBF Ψ uses $O(\log t(n))$ universal variable where n is the length of the input w.*

Note that since the reductions in Theorem 9 and 6 are parsimonious, the algorithm $\mathcal{A}$ in Corollary 10 is also parsimonious in the sense that if $\mathcal{M}$ is the NTM that accepts L, then the number of accepting runs of $\mathcal{M}$ on input word w is the precisely the number of satisfying Skolem functions for the output DQBF Ψ.

▶ **Theorem 11.** *For every language $L \in$ NSPACE$[s(n)]$, there is a deterministic algorithm $\mathcal{A}$ with run time polynomial in n and $s(n)$ such that: On input w, it outputs a 2-DQBF Ψ with $O(s(|w|))$ universal variables such that $w \in L$ if and only if Ψ is not satisfiable.*

Proof. Let $L \in$ NSPACE$[s(n)]$ and let $\mathcal{M}$ be the NTM that accepts L using $s(n)$ space. We may assume that $\mathcal{M}$ halts on every input word. We first present the reduction to 2-CNF formula (with exponential blow-up). On input word w, it construct the following formula, denoted by F_w.

(a) The variables are X_C, where the index C ranges over all the configurations of M on w.

[2] Actually [8, Theorem 1] establishes Theorem 9 for some exponential $t(n)$, i.e., $t(n) = 2^{p(n)}$ for some polynomial $p(n)$. However, it can be easily verified that the proof can be used for Theorem 9.

(b) For every two configurations C_1 and C_2 where C_2 is the next configuration of C_1, we have an implication $X_{C_1} \to X_{C_2}$.

(c) For the initial configuration C_0, we have the implication $\neg X_{C_0} \to X_{C_0}$.

(d) For the initial configuration C_0 and the accepting configuration C_{acc}, we have the implication $X_{C_{acc}} \to \neg X_{C_0}$.

It is not difficult to show that M accepts w if and only if F_w is not satisfiable.

We can easily modify the construction above to the case of DQBF by encoding the configuration of M on w with strings from Σ with length $O(s(n))$, where $n = |w|$. That is, on input word w, we can construct in time polynomial in $s(n)$ a DQBF Ψ such that $\exp(\Psi)$ is equivalent to the formula F_w. The details are as follows. On input word w, construct the following DQBF:

$$\Psi \;:=\; \forall \bar{x}_1 \forall \bar{x}_2\; \exists y_1(\bar{x}_1) \exists y_2(\bar{x}_2) \quad \psi$$

where $|\bar{x}_1| = |\bar{x}_2| = O(s(n))$, i.e., $\bar{x}_1, \bar{x}_2$ are used to represent the 0-1 encoding of the configuration of $\mathcal{M}$ of w and the matrix ψ states the following.

(1) The functions y_1 and y_2 are the same. Formally:

$$\bar{x}_1 = \bar{x}_2 \;\to\; y_1 = y_2$$

(2) If $\bar{x}_1$ and $\bar{x}_2$ encode configurations and $\bar{x}_2$ is the "next" configuration of $\bar{x}_1$, then y_1 implies y_2. Formally:

$$C(\bar{x}_1, \bar{x}_2) \;\to\; (\neg y_1 \;\vee\; y_2)$$

where $C(\bar{x}_1, \bar{x}_2)$ is a formula that checks whether $\bar{x}_1$ and $\bar{x}_2$ are configurations and $\bar{x}_2$ is the next configuration of $\bar{x}_1$. Such formula can be easily constructed in time polynomial in $s(n)$.

(3) If $\bar{x}_1$ is the accepting configuration and $\bar{x}_2$ is the initial configuration, then y_1 implies $\neg y_2$. Formally:

$$C'(\bar{x}_1, \bar{x}_2) \;\to\; (\neg y_1 \;\vee\; \neg y_2)$$

where $C'(\bar{x}_1, \bar{x}_2)$ is a formula that checks whether $\bar{x}_1$ is the accepting configuration and $\bar{x}_2$ is the initial configuration. Such formula can be easily constructed in time polynomial in $\max(n, s(n))$.

(4) If $\bar{x}_1, \bar{x}_2$ are the initial configuration, then $y_1 \;\vee\; y_2$. Formally:

$$C''(\bar{x}_1, \bar{x}_2) \;\to\; (y_1 \;\vee\; y_2)$$

where $C''(\bar{x}_1, \bar{x}_2)$ is a formula that checks whether $\bar{x}_1, \bar{x}_2$ are the initial configuration. Again, such formula can be easily constructed in time polynomial in $\max(n, s(n))$.

It is not difficult to see that on every input word w, the formula F_w is equivalent to $\exp(\Psi)$. This completes the proof of Theorem 11. ◀

It is not difficult to see that if the TM $\mathcal{M}$ in the proof of Theorem 11 is deterministic, we can easily modify the construction such that $w \in L$ if and only if the output Ψ is satisfiable.

6 Concluding remarks

In this paper we have shown that a number of well known classical results on SAT can be lifted up to DQBF. Previously it has been observed that DQBF formulas indeed represent (exponentially large) CNF formulas and that the satisfiability problem becomes NEXP-complete. In this paper we take one little step forward by presenting a few results that shows a strong resemblance between DQBF and SAT. Together with the work in [8], we hope that it can be a convincing evidence for DQBF to be the *bona fide* problem in NEXP.

We believe there is still a lot of work to do. There are still a plethora of results on SAT that we haven't considered and it would be interesting to investigate which results can be lifted to DQBF and which can't. We leave it for future work.

References

1 SAT competition, 2020. URL: `http://www.satcompetition.org/`.

2 V. Balabanov, H.-J. K. Chiang, and J.-H. R. Jiang. Henkin quantifiers and boolean formulae: A certification perspective of DQBF. *Theor. Comput. Sci.*, 523:86–100, 2014.

3 V. Balabanov and J.-H. R. Jiang. Reducing satisfiability and reachability to DQBF. In *Talk given at QBF*, 2015.

4 A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. IOS Press, 2009.

5 R. Bloem, R. Könighofer, and M. Seidl. SAT-based synthesis methods for safety specs. In *VMCAI*, 2014.

6 U. Bubeck. *Model-based transformations for quantified boolean formulas*. PhD thesis, University of Paderborn, 2010.

7 K. Chatterjee, T. Henzinger, J. Otop, and A. Pavlogiannis. Distributed synthesis for LTL fragments. In *FMCAD*, 2013.

8 F.-H. Chen, S.-C. Huang, Y.-C. Lu, and T. Tan. Reducing NEXP-complete problems to DQBF. In *FMCAD*, 2022.

9 D. Chistikov, C. Haase, Z. Hadizadeh, and A. Mansutti. Higher-order quantified boolean satisfiability. In *MFCS*, 2022.

10 S. Cook. The complexity of theorem proving procedures. In *STOC*, 1971.

11 S. Cook. Short propositional formulas represent nondeterministic computations. *Inf. Process. Lett.*, 26(5):269–270, 1988.

12 A. Fröhlich, G. Kovásznai, and A. Biere. A DPLL algorithm for solving DQBF. In *POS-12, Third Pragmatics of SAT workshop*, 2012.

13 A. Fröhlich, G. Kovásznai, A. Biere, and H. Veith. iDQ: Instantiation-based DQBF solving. In *POS-14, Fifth Pragmatics of SAT workshop*, 2014.

14 H. Galperin and A. Wigderson. Succinct representations of graphs. *Inf. Control.*, 56(3):183–198, 1983.

15 A. Ge-Ernst, C. Scholl, and R. Wimmer. Localizing quantifiers for DQBF. In *FMCAD*, 2019.

16 Aile Ge-Ernst, Christoph Scholl, Juraj Síc, and Ralf Wimmer. Solving dependency quantified boolean formulas using quantifier localization. *Theor. Comput. Sci.*, 925:1–24, 2022.

17 K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker. Equivalence checking of partial designs using dependency quantified boolean formulae. In *ICCD*, 2013.

18 K. Gitina, R. Wimmer, S. Reimer, M. Sauer, C. Scholl, and B. Becker. Solving DQBF through quantifier elimination. In *DATE*, 2015.

19 J.-H. R. Jiang. Quantifier elimination via functional composition. In *CAV*, 2009.

20 R. Jiang. Second-order quantified boolean logic. In *AAAI*, 2023.

21 G. Kovásznai. What is the state-of-the-art in DQBF solving. In *Join Conference on Mathematics and Computer Science*, 2016.

22 A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 21(12):1377–1394, 2002.

23 O. Kullmann and A. Shukla. Autarkies for DQCNF. In *FMCAD*, 2019.

24 L. Levin. Universal search problems (in russian. *Problems of Information Transmission*, 9(3):115–116, 1973.

25 C. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Inf. Control.*, 71(3):181–185, 1986.

26 G. Peterson and J. Reif. Multiple-person alternation. In *FOCS*, 1979.

27 F.-X. Reichl and F. Slivovsky. Pedant: A certifying DQBF solver. In *SAT*, 2022.

28 F.-X. Reichl, F. Slivovsky, and S. Szeider. Certified DQBF solving by definition extraction. In *SAT*, 2021.

29 C. Scholl and B. Becker. Checking equivalence for partial implementations. In *DAC*, 2001.

30 C. Scholl, R. J.-H. Jiang, R. Wimmer, and A. Ge-Ernst. A PSPACE subclass of dependency quantified boolean formulas and its effective solving. In *AAAI*, 2019.

31 C. Scholl and R. Wimmer. Dependency quantified boolean formulas: An overview of solution methods and applications - extended abstract. In *SAT*, 2018.

32 J. Síc and J. Strejcek. DQBDD: an efficient bdd-based DQBF solver. In *SAT*, 2021.

33 S. Skyum and L. Valiant. A complexity theory based on boolean algebra. *J. ACM*, 32(2):484–502, 1985.

34 L. Stockmeyer and A. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, 1973.

35 L. Tentrup and M. Rabe. Clausal abstraction for DQBF. In *SAT*, 2019.

36 K. Wimmer, R. Wimmer, C. Scholl, and B. Becker. Skolem functions for DQBF. In *ATVA*, 2016.

37 R. Wimmer, A. Karrenbauer, R. Becker, C. Scholl, and B. Becker. From DQBF to QBF by dependency elimination. In *SAT*, 2017.

38 R. Wimmer, S. Reimer, P. Marin, and B. Becker. HQSpre – an effective preprocessor for QBF and DQBF. In *TACAS*, 2017.

39 R. Wimmer, C. Scholl, and B. Becker. The (D)QBF preprocessor hqspre - underlying theory and its implementation. *J. Satisf. Boolean Model. Comput.*, 11(1):3–52, 2019.

Effective Auxiliary Variables via Structured Reencoding

Andrew Haberlandt[1]
Carnegie Mellon University, Pittsburgh, PA, USA

Harrison Green[1]
Carnegie Mellon University, Pittsburgh, PA, USA

Marijn J. H. Heule
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Extended resolution shows that auxiliary variables are very powerful in theory. However, attempts to exploit this potential in practice have had limited success. One reasonably effective method in this regard is bounded variable addition (BVA), which automatically reencodes formulas by introducing new variables and eliminating clauses, often significantly reducing formula size. We find motivating examples suggesting that the performance improvement caused by BVA stems not only from this size reduction but also from the introduction of *effective auxiliary variables*. Analyzing specific packing-coloring instances, we discover that BVA is *fragile* with respect to formula randomization, relying on variable order to break ties. With this understanding, we augment BVA with a heuristic for breaking ties in a *structured* way. We evaluate our new preprocessing technique, Structured BVA (SBVA), on more than 29 000 formulas from previous SAT competitions and show that it is robust to randomization. In a simulated competition setting, our implementation outperforms BVA on both randomized and original formulas, and appears to be well-suited for certain families of formulas.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Reencoding, Auxiliary Variables, Extended Resolution

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.11

Supplementary Material *Software*: `https://github.com/hgarrereyn/SBVA`
archived at `swh:1:dir:b202295533cc92ae2d5fc33c51bd5eeccd530d2a`

Funding The authors are supported by National Science Foundation grant CCF-2015445.
Andrew Haberlandt: NSF Graduate Research Fellowship Grant No. DGE-2140739.

Acknowledgements The authors thank Bernardo Subercaseaux for his assistance revising this paper, and the Pittsburgh Supercomputing Center for allowing us to use the Bridges2 [7] cluster for our experiments.

1 Introduction

Pre-processing techniques that introduce and eliminate auxiliary variables have been shown to be helpful both in theory and practice. Theoretically, auxiliary variables lift the power of solvers from the resolution proof system to Extended Resolution (ER) [23, 8, 11]. In practice, efforts to exploit this full power of ER have had limited success; however, auxiliary variables have been used to reencode formulas in a way that drastically reduces their size [4, 2, 15], often leading to a decreased solve time. In this work we show that this speedup may not be caused entirely by the reduction in formula size, but by the introduction of certain *effective auxiliary variables*.

[1] Authors contributed equally.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 11; pp. 11:1–11:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A very powerful pre-processing technique is *Bounded Variable Elimination* (BVE) [9]. As its name suggests, it eliminates a variable x by resolving each clause containing a literal x with every clause containing literal $\overline{x}$. Importantly, BVE only performs such an elimination if it helps reduce the formula size (measured as the number of clauses plus the number of variables). A pre-processing technique that introduces new auxiliary variables is *Bounded Variable Addition* (BVA) [15], which is the focus of this article. It is well known that the introduction of auxiliary variables is crucial for many succinct encodings (e.g., the Tseitin transformation [23], or cardinality constraints [20, 14]). Following the intuition of BVE, BVA will only introduce a new variable if it can eliminate a larger number of clauses than it adds.

Auxiliary variables may not only be useful to reduce the size of a formula, but they can also capture some semantic meaning about the underlying problem to encode, as we detail in Section 3. As a case study, we consider a recent encoding used by Subercaseaux and Heule for computing the packing chromatic number of the infinite square grid via SAT solving [22]. In their work, BVA was found to generate auxiliary variables that represented clusters of neighboring vertices of the grid. The encoding resulting from running BVA on a direct encoding of the problem inspired a more efficient encoding, by suggesting the usefulness of having auxiliary variables capturing clusters of vertices. In this paper, we offer new insight into this *"meaningful variables"* phenomenon, which we believe can generalize to other problems as well. Furthermore, even though the reencoding resulting from BVA suggested meaningful new variables for the packing coloring problem, it was not as effective as manually designing a more structured encoding based on some of those variables. We take this as motivation to identify shortcomings of BVA and improve upon its design.

In general, on problems where BVA is effective, the effect tends to be extreme. BVA is able to reduce the number of clauses by a ×10 factor or more, improving solve time by orders of magnitude. However, in this paper, we find that this reduction in solve time is highly sensitive to randomly scrambling the formula (even when controlling for how CDCL solvers are generally sensitive to this form of randomization [3]). In particular, randomizing the order of variables and clauses prior to BVA substantially reduces the positive effect of BVA on solve time, despite maintaining the same overall reduction in formula size. Using the packing k-color problem, we show that the effectiveness of BVA relies on the introduction of a few specific variables that account for only a small fraction of the reduction in formula size. Moreover, we identify that the lack of effective tie-breaking in BVA is the cause of this high sensitivity to randomization. Inspired by these new insights into the behavior of BVA, we present SBVA (Section 4), a version augmented with a tie-breaking heuristic that enables it to introduce better auxiliary variables at each step, even when the original formula is randomized. Our heuristic is based on a connectivity measure between variables in the *incidence graph* of CNF formulas, which is preserved under randomization of the formula. As a result, SBVA is able to identify effective auxiliary variables even when the original formula is scrambled. We evaluate our implementation by running it on more than 29 000 formulas from the Global Benchmark Database [12]. Experimental results, presented in Section 5, demonstrate that our approach outperforms the original implementation of BVA.

In summary, the main contributions of this article are:

1. We offer new insight into the behavior of BVA, by exhibiting its ability to introduce effective auxiliary variables and showing its sensitivity to formula randomization.
2. We design SBVA, a heuristic-guided form of BVA, that introduces new variables in a way that is robust to randomization.
3. We perform a large-scale evaluation of both BVA and SBVA on benchmark problems from the SAT Competition and study their behavior on different families of instances.
4. We release an open-source implementation of SBVA that supports DRAT proof logging.

2 Preliminaries

A *literal* is either a variable x, or its negation ($\overline{x}$). A *propositional formula* in *conjunctive normal form* (CNF) is a conjunction of *clauses*, which are themselves disjunctions of literals. An *assignment* is a mapping from variables to truth values. A positive (negative) literal is true if the corresponding variable is assigned to true (false, respectively). An assignment satisfies a clause if at least one of its literals is true, and we say an *formula* is satisfied if all of its clauses are. A formula is *satisfiable* (SAT) if there exists an assignment that satisfies it, or *unsatisfiable* (UNSAT) otherwise. For example, the formula $(x \vee \overline{y}) \wedge (\overline{x} \vee z)$ is made up of two clauses, $(x \vee \overline{y})$ and $(\overline{x} \vee z)$, each with two literals. This formula is satisfiable, since the assignment of x and z to true and y to false satisfies it.

Auxiliary Variables. There can be many equivalent ways of encoding a problem into CNF, differing in the meaning assigned to individual variables. Problems often have a *direct encoding*, in which variables are assigned for each individual decision element present in a problem. For example, in a direct encoding of graph coloring, there are $k|V|$ variables, where each $v_{i,c}$ represents whether node i has color c and k is the number of colors.

Although direct encodings are often the most intuitive, more efficient encodings are known for a wide variety of problems. These encodings often add *auxiliary variables* to the formula, which capture properties about a group of variables. One of the simplest examples is an AtMostOne$(x_1, \ldots, x_n)$ constraint, which requires that at most one of the variables $x_1, \ldots, x_n$ is true. Without adding auxiliary variables, this constraint requires $\Theta(n^2)$ clauses, which are typically binary clauses between every pair of variables [14]. However, with the introduction of auxiliary variables, this constraint can be encoded in a linear number of clauses and variables as follows [13]:

$$\text{AtMostOne}(x_1, \ldots, x_n) = \text{AtMostOne}(x_1, x_2, x_3, y) \wedge \text{AtMostOne}(x_4, \ldots, x_n, \overline{y}) \tag{1}$$

where the pairwise encoding is used for AtMostOne$(x_1, \ldots, x_n)$ where $n < 4$. The split AtMostOne constraints require that at most one of $\{x_1, x_2, x_3\}$ is true, and at most one of $\{x_4, \ldots, x_n\}$ is true, respectively. The added auxiliary variable y prevents a variable in both of the groups from being true. The auxiliary variable y is forced false if any of x_1, x_2, or x_3 are true, and forced true if any of $x_4, \ldots, x_n$ are false. If a literal from both groups is true, the auxiliary variable y prevents the formula from being satisfiable.

Extended Resolution. Starting from the original formula, the Extended Resolution proof allows only two simple rules:

1. *Resolution*: Given clauses $C \vee p$ and $D \vee \overline{p}$, add the clause $C \vee D$ to the proof.
2. *Extension*: Define a new variable x as $x \leftrightarrow \overline{a} \vee \overline{b}$, where a and b are literals in the current proof. Add the clauses $x \vee a$, $x \vee b$, and $\overline{x} \vee \overline{a} \vee \overline{b}$ to the proof.

In *resolution*, the clause $C \vee D$ is implied by the first two clauses, resulting in a logically equivalent formula. In *extension*, however, the introduction of a new variable x is not implied by the original clauses, and results in a formula that preserves satisfiability and is only logically equivalent over the original variables.

Using the extension rule, new variables can be defined in terms of existing variables. The original rule defined by Tseitin [23] only allows for definitions of the form $x \leftrightarrow \overline{a} \vee \overline{b}$, the construction for which is given in the definition above. However, the extension rule can be applied repeatedly to construct variables corresponding to arbitrary propositional formulas over the original variables. This flexibility is key to the success of Extended Resolution, but it provides no guidance on how these extensions should be chosen.

Bounded Variable Addition. Bounded Variable Addition (BVA) [15] is a pre-processing technique that reduces the number of clauses in a formula by adding new variables. Each application of BVA first identifies a "grid" of clauses, as shown in Figure 1. Then, BVA adds a new variable and clauses which resolve together to generate all clauses in the grid.

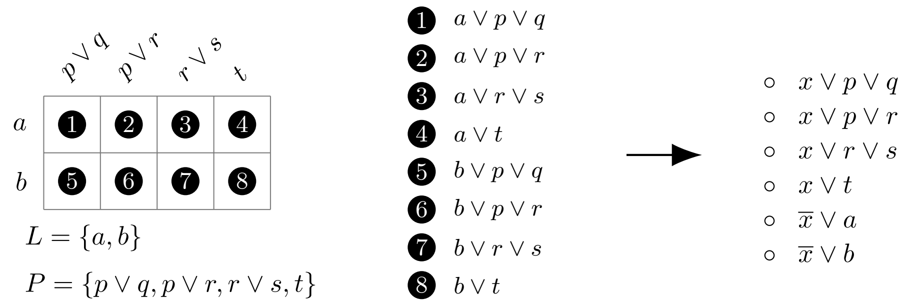


Figure 1 Bounded variable addition transforms groups of clauses (those that form a grid) by adding a new variable and eliminating a number of clauses.

Collectively, for a formula F, the grid constitutes a set of literals L and a set of partial clauses P, such that $\forall l \in L, \forall C \in P : (l \lor C) \in F$. While bounded-variable elimination eliminates variables by replacing all the clauses containing a variable with their resolvents, BVA tries to identify grids of *resolvents* which can be generated by the introduction of a new variable and a smaller number of clauses. These grids of clauses capture the fact that either the entirety of L must be satisfied, or the entirety of P must be satisfied. More precisely, $F \implies (\bigwedge_{l \in L} l) \lor (\bigwedge_{C \in P} C)$. By identifying these grids, BVA replaces $|L| \cdot |P|$ clauses with a single, new variable x and $|L| + |P|$ clauses (which can generate the original set by resolution on x in $\{x \lor C \mid C \in P\} \times \{\overline{x} \lor l \mid l \in L\}$). Therefore, if $|L| \cdot |P| > |L| + |P| + 1$, then this replacement results in a reduction in formula size.

Note that a BVA replacement step can be simulated by extended resolution: First, add the definition $x \leftrightarrow \text{AND}(L)$. In the example above, this means adding the clauses $x \lor \overline{a} \lor \overline{b}$, $\overline{x} \lor a$, and $\overline{x} \lor b$. Afterwards, the clauses $x \lor p \lor q$, $x \lor p \lor r$, $x \lor r \lor s$, and $x \lor t$ can each be derived using $|L|$ resolution steps. For example, to derive $x \lor p \lor q$, resolve $x \lor \overline{a} \lor \overline{b}$ with $a \lor p \lor q$ and the result with $b \lor p \lor q$. Afterward the clauses used in these resolution steps can be deleted.

The SimpleBoundedVariableAddition **algorithm.** Manthey et al. [15] propose a greedy algorithm to identify these grids of resolvents that prioritizes literals which appear in many clauses called SimpleBoundedVariableAddition. An abbreviated version of a single variable addition in this algorithm is shown in Algorithm 1.

Each identified grid starts from the most frequently occurring literal l in the current formula. The grid starts with dimension $1 \times |F_l|$, where F_l is the set of clauses containing l. From there, the algorithm searches for a literal $l_{\max}$ to add to the grid, which maximizes the number of remaining resolvents.

To identify the literal $l_{\max}$, the BVA algorithm looks for the literal for which $(l_{\max} \lor C)$ appears in F for the greatest number of clauses $C \in P$ (line 4). At each step, a literal is added to L (line 6), and clauses may be removed from P (line 7). The grid will continue to shrink until the addition of a literal to the grid would not increase the size of the formula reduction (line 5), as shown in Figure 2.

In BVA, variable additions are performed as long as there is a reduction in formula size.

Algorithm 1 A single variable addition in SimpleBoundedVariableAddition [15].

```
$\textsc{PartialClauses}(F,\, l) := \{C \setminus \{l\} \mid (C \in F) \wedge (l \in C)\}$
$F :=$ the clauses in the current formula
$l :=$ a literal in $F$

$L \leftarrow \{l\}$
$P \leftarrow \textsc{PartialClauses}(F, l)$
while True do
    $l_{\max} = \operatorname{argmax}_{l_m \in \text{Lits}(F)} |P \cap \textsc{PartialClauses}(F, l_m)|$    ▷ Sensitive to tiebreaking
    if adding $l_{\max}$ results in a greater reduction then
        $L \leftarrow L \cup \{l_{\max}\}$
        $P \leftarrow P \cap \textsc{PartialClauses}(F, l_{\max})$
    else
        break
if $|L| \cdot |P| > |L| + |P| + 1$ then    ▷ If adding this variable would reduce the formula size
    $S_{\text{add}} \leftarrow \{x \vee C \mid C \in P\} \cup \{\overline{x} \vee l_m \mid l_m \in L\}$    ▷ Introduce a new variable $x$
    $S_{\text{remove}} \leftarrow \{l_i \vee C \mid (l_i, C) \in L \times P\}$
    $F \leftarrow (F \setminus S_{\text{remove}}) \cup S_{\text{add}}$
```

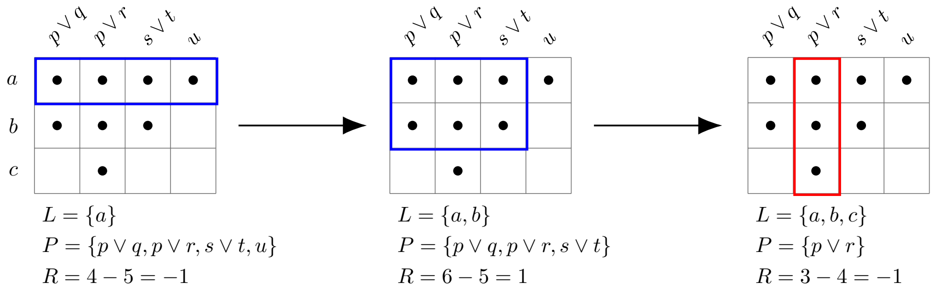


Figure 2 BVA adds variables to form a grid, until the reduction stops increasing. Here, the largest reduction was 1, and the variable corresponding to the middle grid will be added.

The entirety of Algorithm 1 is repeated using different literals for l to construct multiple new auxiliary variables. Specifically, the original implementation defines a priority queue of literals ordered by the number of clauses each literal appears in. Our adaptation of BVA (Section 4) reuses this implementation detail. These repeated applications of BVA enable the algorithm to achieve large reductions in formula size, and auxiliary variables added in previous steps can even be re-used in future variable introductions.

3 Motivating Example

To motivate the need for a heuristic-guided version of BVA, we will first demonstrate the effect of randomization on existing implementations of BVA, and the disproportionate impact of a few critical variable additions.

3.1 Packing Colorings

BVA has been shown to be effective on the grid packing k-coloring problems, whose constraints are based on coloring a circular grid of tiles, shown in Figure 3a. Unlike a standard graph coloring problem, each color in the packing k-coloring problem is associated with a integer distance from 1 to k. When coloring the grid, two tiles can only have the same color if the taxicab distance between them is greater than the color number. For example, two tiles of color 3 must have at least 3 tiles between them, while color 1 tiles cannot be adjacent. The $D_{r,k}$ problem asks whether the grid of radius r can be colored with k colors.

The direct encoding for this problem consists of variables $v_{i,c}$, denoting that grid location i has color c. There are three types of clauses [22]:

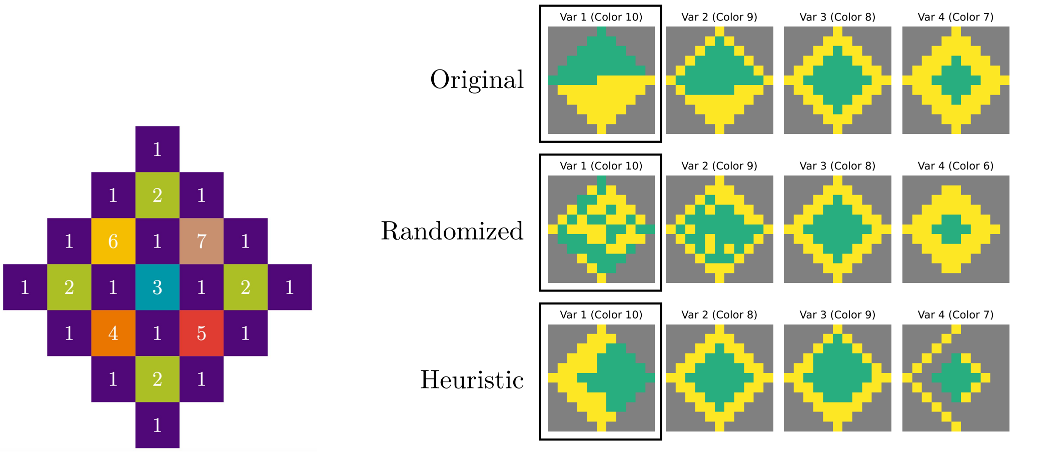
1. At-Least-One-Color: $\forall i, (v_{i,1} \vee v_{i,2} \vee \cdots \vee v_{i,k})$. Each tile must be colored with a color between 1 and k.
2. At-Most-One-Distance: $\forall i, j, c : d(i, j) \leq c, (\overline{v_{i,c}} \vee \overline{v_{j,c}})$. If the distance between two tiles is less than or equal to the color, they cannot both have that color.
3. Center-Clause: $v_{(0,0),c}$ for a fixed color c. This is a symmetry-breaking optimization [21], which has no effect on BVA since it ignores unit clauses.

Previous work [22] showed that BVA can reduce the size of such formulas by a factor of 4, and induces more than a ×4 speedup on the larger instance ($D_{6,11}$). They found that auxiliary variables capture *regions* of grid tiles within a particular color, i.e. the grid replacement happens entirely within the binary at-most-one-distance constraints.

We visualize the variables introduced by BVA on $D_{5,10}$, the packing k-coloring problem with radius 5 and 10 colors. In the first row of Figure 3b, each of the four plots introduces a new auxiliary variable x for one of the colors $c \in \{1, \ldots, 10\}$ (denoted above each plot). All the binary clauses for color c (At-Most-One-Distance clauses) of the form $(\overline{v_{i,c}} \vee \overline{v_{j,c}})$ with grid location i corresponding to a green square and grid location j corresponding to a yellow square will be replaced with a smaller number of clauses: $(x \vee \overline{v_{i,c}})$ for each green location i and $(\overline{x} \vee \overline{v_{j,c}})$ for each yellow location j.

3.2 Negative Impact of Randomization

We discovered that randomizing packing k-coloring formulas prior to running BVA significantly increases the resulting solve time. Furthermore, the variables added by BVA after randomization fail to capture the clustered *regions* within the problem's 2D space that are



(a) Figure from [22] showing the $D_{3,5}$ grid packing k-coloring problem

(b) The effect of variable randomization on the first four BVA substitutions in $D_{5,10}$. The black boxes indicate the first variable addition, the effect of which is isolated in Table 1.

Figure 3 The auxiliary variables introduced by BVA on the packing k-coloring problem are sensitive to randomization.

Table 1 CaDiCaL solve time for the $D_{5,10}$ packing problem, breaking BVA ties using the original variable order (sorted), randomized variable order (randomized), or the heuristic proposed in Section 4 (heuristic). Breaking ties differently has a *significant* effect on solve time even when the resulting formula is the same size (Single BVA Step).

	# Vars	# Clauses	Solve (s)
Original formula	610	10688	590.545
Single BVA Step (sorted)	611	9819	105.635
Single BVA Step (randomized)	611	9819	429.396
Single BVA Step (heuristic, **this paper**)	611	9819	213.018
Full BVA (sorted)	973	2313	38.749
Full BVA (randomized)	971	2305	107.675
Full BVA (heuristic, **this paper**)	972	2290	55.482

identified without randomization. Figure 3b shows the first four variable additions performed by BVA on $D_{5,10}$. The effect is especially noticeable in the first few variable additions. The structure of these variables is more than a visual artifact. Running BVA to completion produces a formula that requires more than ×2 the solve time in CaDiCaL compared with running BVA on the original formula, despite a similar reduction in formula size (see Table 1).

We found that the *first* variable added by BVA in $D_{5,10}$ had a disproportionate impact on the solve time of the formula. We isolated the effect of a single replacement by allowing BVA to only produce one new auxiliary variable and then evaluating the solve time of the resulting formula. Table 1 shows that a single variable addition (outlined in black in Figure 3b) can achieve a ×6 speedup over the original formula and that the impact of this single addition is also substantially affected by randomization. Although randomization before BVA did not affect the size reduction of the first variable addition, the randomized formula with a single BVA step is 2 times slower compared to the original formula.

The importance of individual variable additions and their sensitivity to randomization suggests that BVA's impact is derived not only from the size reduction but from the *structure* of the variable additions.

3.3 Ties in Bounded Variable Addition

The reason for the BVA's sensitivity to randomization is due to a detail in the way implementations treat ties between literals. As described in Section 2, the algorithm chooses the literal that maximizes the number of remaining resolvents to be eliminated (Algorithm 1, line 4). If there is a tie between two literals, the original algorithm does not specify which literal should be used. The original implementation provided by [15] breaks ties using the variable number in the original formula. Figure 4 shows how breaking ties differently leads to different variable additions. Note that since BVA eliminates the clauses in the grid when adding a variable, it is *not* possible for multiple applications of BVA to eventually add both variables resulting from a tie.

In the $D_{5,10}$ packing problem, colors 9 and 10 are almost fully connected; coloring a tile with color 10 means that no tile within 10 spaces of it can also be colored 10. When BVA creates a variable for these pairwise constraints, all of the clauses are tied for the number of preserved resolvents (since every pair of color-10 variables appears in a at-most-one-distance clause). Since the original implementation used variable number to break ties

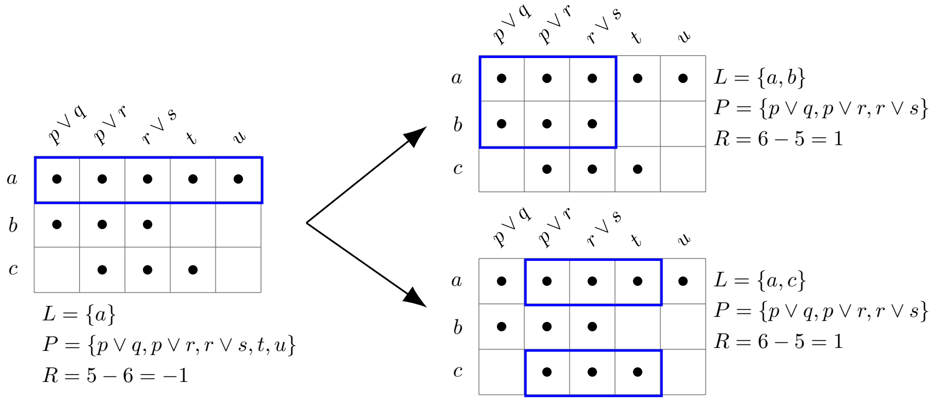


Figure 4 The addition of b or c both lead to a 2×3 grid of resolvents. Breaking this tie in different ways leads to different variable additions.

and ordered variables from top-left to bottom-right, the variable additions it produces follow that structured pattern. However, when the variable order is randomized, the resulting region lacks structure and the formula takes longer to solve.

3.4 Recovering Structure

After randomization, BVA struggles to introduce variables that represent coherent clusters of tiles. However, we note that the original structure is still captured by the original formula as a whole. For example, in the $D_{5,10}$ packing problem, two variables representing color 1, $v_{i,1}$ and $v_{j,1}$, only share a pairwise constraint if they are adjacent (i.e. if i and j represent adjacent tiles). If we could recover a generic metric for how *close* variables are to each other (e.g. in the 2D space of $D_{5,10}$), this metric could be used to help BVA recover structure in problems where the original variable order does not result in structured variable additions.

The intuition for our heuristic, which is detailed in Section 4, is based on the structure observed in the packing problem. We notice that while variables in color 10 are indistinguishable after randomization (i.e. all fully connected with At-Most-One-Distance clauses), the variables in color 1 preserve the structure of the original problem: these variables only share At-Most-One-Distance clauses with their immediate neighbors. Additionally, variables for the same *tile location* but different *colors* are all linked by an At-Least-One-Color constraint, even after randomization. One could deduce which variables in color 10 are neighbors by looking at the connectivity of the equivalent tile positions in color 1. Specifically this requires 3 "hops" through clauses: starting at a variable $v_{i,10}$ in color 10, we find $v_{i,1}$ in color 1 (via an At-Least-One-Color clause), then find $v_{j,1}$ in color 1 (via an At-Most-One-Distance clause), and finally find $v_{j,10}$ in color 10 (via an At-Least-One-Color clause); the full path is $v_{i,10} \rightarrow v_{i,1} \rightarrow v_{j,1} \rightarrow v_{j,10}$.

While it is possible to construct an algorithm to recover this structure specifically for the k-coloring packing problem, we generalize this concept by *counting* paths. Specifically, between two variables $v_{i,10}$ and $v_{j,10}$ in color 10 there are *many* paths of length 3: for example $v_{i,10} \rightarrow v_{a,10} \rightarrow v_{b,10} \rightarrow v_{j,10}$ (using only At-Most-One-Distance clauses). However, only *adjacent* variables in color 10 will have the additional path that goes through color 1: $v_{i,10} \rightarrow v_{i,1} \rightarrow v_{j,1} \rightarrow v_{j,10}$. For a given variable in color 10, it will have the most 3-hop paths to variables of the immediately adjacent grid tiles. We formalize this intuition in Section 4.

4 Structured Reencoding

In this section we define our implementation of a heuristic for breaking ties during variable selection in BVA. While our heuristic was initially designed to mitigate the detrimental effects of randomization on the packing coloring problems, we found that it is also effective for other problems, even ones which have not been randomized. In Section 5 we show that our heuristic-guided BVA is effective on a wide variety of problems and offers a significant improvement to solve time for certain families of formulas.

The 3-Hop Heuristic. Our heuristic is based on the intuition that BVA should prefer to break ties by adding variables that are *close* to one another. In Subsection 3.4, we noticed that in the k-coloring problem, there are some paths between variables that are only present when variables are *close* in the problem's 2-D space. The *variable incidence graph* compactly captures this notion of variable adjacency. Here we formally define a heuristic for "variable distance" based on the number of *paths* between pairs of variables in the *variable incidence graph.*

▶ **Definition 1.** *The Variable Incidence Graph (VIG) of a formula F is an undirected graph $G = (V, E)$ where V is the set of variables in F, and E contains an edge between variables if they appear in a clause together. The weight on an edge (v_1, v_2) is the number of clauses in which v_1 and v_2 appear together:* $w(v_1, v_2) = |\{C \in F : \{v_1, v_2\} \subset \mathrm{Vars}(C)\}|$

We measure variable distance by counting the number of distinct paths between two variables (i.e. using different intermediate variables or clauses). Edges in the VIG indicate the number of clauses shared by pairs of variables. For a given sequence of variables $(v_1, v_2, ..., v_n)$ the number of distinct paths through different combinations of clauses is given by $w(v_1, v_2) \cdot w(v_2, v_3) \cdot \ldots \cdot w(v_{n-1}, v_n)$. Since edge weights are multiplicative along a path, the number of different paths of length n through the VIG is given by A^n, where A is the adjacency matrix of the VIG. Since we identified that adjacent tiles in the packing problem have more length-3 paths between them, we define a simple heuristic that counts the number of paths of length 3 in the VIG, which we call the *3-hop heuristic.*

▶ **Definition 2.** *The 3-hop heuristic $H(x, y)$ is defined as the number of distinct paths of length 3 between two variables x and y in the VIG. Two paths are distinct if they travel through a different sequence of variables or clauses. Given the VIG adjacency matrix A, the 3-hop heuristic can be computed as $H(x, y) = (A^3)_{x,y}$.*

We modify Algorithm 1 to use our heuristic as a tie-breaker, specifically augmenting the computation of argmax in line 4: when multiple values of l_m have the same number of remaining resolvents, we choose the literal l_m with the highest value of $H(l, l_m)$. Our implementation of BVA, called SBVA, is written in C++ and uses the `Eigen` library for sparse matrix operations. It is capable of generating DRAT proofs describing the sequence of variable additions and clause deletions and thus could be used with a solver to generate certificates of unsatisfiability.

In Figure 5, we show the value of $H(x, y)$ in $D_{5,10}$ for variables representing color 10 between a variable of interest (outlined in black) and all other variables of color 10. Grid tiles that are closer in the 2-D space of the packing k-coloring problem have more 3-hop paths between them and thus have a higher heuristic value. Using our heuristic on a randomized formula for the packing problem, we recover variables that capture the spatial structure of the problem. In the third row of Figure 3b, we show the first 4 variables added by SBVA,

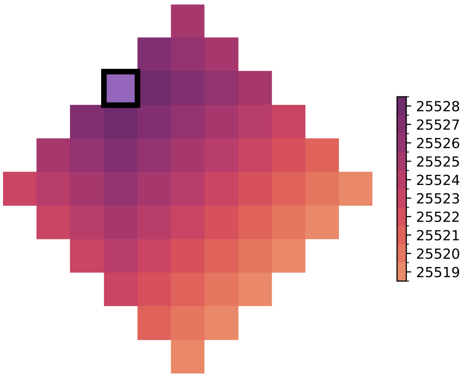


Figure 5 The value of the 3-hop heuristic in $D_{5,10}$ between the color-10 variable for the location outlined in black and all other color-10 locations.

which cluster variables together using the notion of distance that is inherent in the original problem. Furthermore, we find that applying this heuristic to the packing problem results in formulas that solve much faster than BVA on a randomized formula (Table 1).

5 Experimental Details

We evaluated BVA on more than 29,000 formulas from the Global Benchmark Database [12] in order to study the effects of randomization and our heuristic on BVA. In this section, we discuss the experimental setup and provide a brief overview of the results. In Section 6, we analyze the results in more detail and discuss families of formulas that were significantly impacted by BVA and/or SBVA.

Configurations. We constructed three solver configurations that use BVA in different ways. All three variants take a formula, (optionally) randomize it with `scranfilize`, run BVA (with or without heuristic), and pass it to CaDiCaL to solve. For comparison, we include a baseline variant that does not run BVA. Since the particular ordering of clauses and variables in a formula can impact solver performance [3], we also use the `scranfilize` tool immediately prior to running CaDiCaL in all configurations. To mitigate this variance, we run the entire sequence three times for each configuration, averaging across the three runs. The list of configurations is shown in Table 2. Note that all four configurations have randomization applied prior to solving with CaDiCaL but only BVA-RAND-ORIG and BVA-RAND-3HOP have randomization applied prior to BVA/SBVA.

Table 2 Experimental configurations. *Pre* and *Post* refer to arguments passed to `scranfilize` before and after running the preprocessor respectively. An empty space indicates the step was skipped for this variant.

Variant	Pre	Preprocessor	Post	Solver
BASELINE			`-p -P -f 0.5`	CaDiCaL
BVA-ORIG		BVA	`-p -P -f 0.5`	CaDiCaL
BVA-RAND-ORIG	`-p -P -f 0.5`	BVA	`-p -P -f 0.5`	CaDiCaL
BVA-RAND-3HOP	`-p -P -f 0.5`	SBVA	`-p -P -f 0.5`	CaDiCaL

Table 3 PAR-2 scores and number of formulas solved for each variant split by problem type (ALL/UNSAT/SAT) and dataset (FULL/ANNI-2022). Bold cells indicate the lowest PAR-2 score or highest number solved for that group.

		ALL		UNSAT		SAT	
Dataset	Variant	PAR-2	#	PAR-2	#	PAR-2	#
FULL	Baseline	1077.91	21602	756.14	6495	1196.99	15107
	BVA-orig	867.04	22140	**635.71**	6562	948.85	15578
	BVA-rand-orig	870.20	22077	673.58	6533	953.25	15544
	BVA-rand-3hop	**862.29**	**22173**	650.41	**6568**	**935.38**	**15605**
ANNI-2022	Baseline	1262.18	3953	1164.61	2048	**1309.41**	1905
	BVA-orig	**1174.80**	3987	**967.85**	2085	1338.31	1902
	BVA-rand-orig	1193.27	3958	1053.75	2060	1350.09	1898
	BVA-rand-3hop	1188.63	**3995**	982.84	**2088**	1350.98	**1907**

Benchmarks. We evaluated our variants on 29 402 benchmark instances (downloaded on February 20, 2023) from the Global Benchmark Database (GBD) [12]. We also report results against the Anniversary Track from the SAT Competition 2022 [1] (labeled as "ANNI-2022" within this paper) which is included as a subset in the GBD (5355 benchmarks).

Hardware. All experiments were performed on the Bridges-2 system at the Pittsburgh Supercomputing Center [7] on nodes with 128 cores and 256 GB RAM.

Experimental Setup. We compare the four configurations in a simulated competition setting with a fixed time limit of 5 000 seconds per benchmark. The total time is computed as the sum of BVA and CaDiCaL runtimes (scranfilize time is not counted towards this limit). As noted by Manthey et al. [15], BVA can be quite expensive, even on formulas that do not reduce significantly. We allow all versions of BVA to run for 200 seconds and if it has not terminated by then, we instead run the original formula with CaDiCaL. On our full benchmark, BVA terminates within 200 seconds on approximately 95% of problems. We ran 128 instances in parallel per node, leaving approximately 2GB of memory (for reference, in the SAT Competition 2022 [1], solvers were allotted 128GB) for each BVA/CaDiCaL process. This limit is enough for most formulas, but in cases where BVA runs out of memory, we instead run the original formula in CaDiCaL. In both cases (timeout and out-of-memory), the *already-used* time is added to the subsequent solve time of the original formula. This setup provides a fair comparison as BVA could be realistically configured this way in a competition.

We report the PAR-2 scores and number of formulas solved for each variant in Table 3. The PAR-2 score is computed as the total time it took to solve an instance (BVA runtime + CaDiCaL runtime) or twice the time limit if the formula was not solved within 5 000 seconds. We compute the PAR-2 score individually for each run and average across the three runs of a given formula. A formula is marked as solved in Table 3 if any of the three runs solved it within the time limit. Additionally, the set of formulas over which PAR-2 is computed consists of instances where at least one of the four configurations was able to solve it. Instances that were not solved by any configuration were not included in the PAR-2 score. Adding these entirely unsolved instances would not change the number solved and would simply scale the PAR-2 scores equally for all configurations.

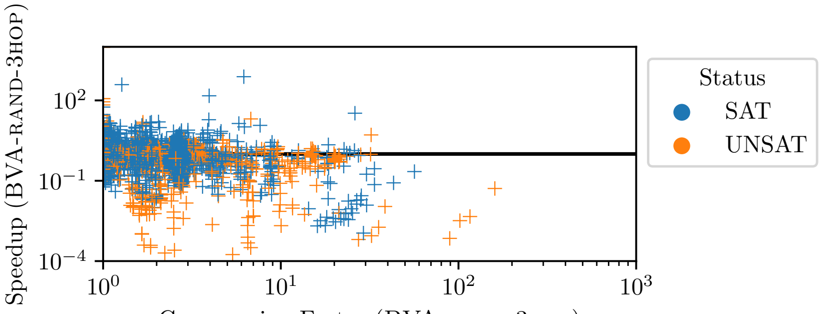


Figure 6 Formula speedup compared to compression factor for BVA-RAND-3HOP.

6 Results and Analysis

This section takes a closer look at the performance of BVA-ORIG, BVA-RAND-ORIG, and BVA-RAND-3HOP in comparison to the BASELINE configuration. We explore both the effects of randomization and the effects of the heuristic, in general and on specific families of formulas. Specifically, we explore the following questions:

Q1: Does compression factor correlate with solve time in the context of BVA?
Q2: What is the effect of randomization on the performance of BVA?
Q3: Can our heuristic outperform randomized BVA?
Q4: How does the performance of our heuristic vary across different families of formulas?

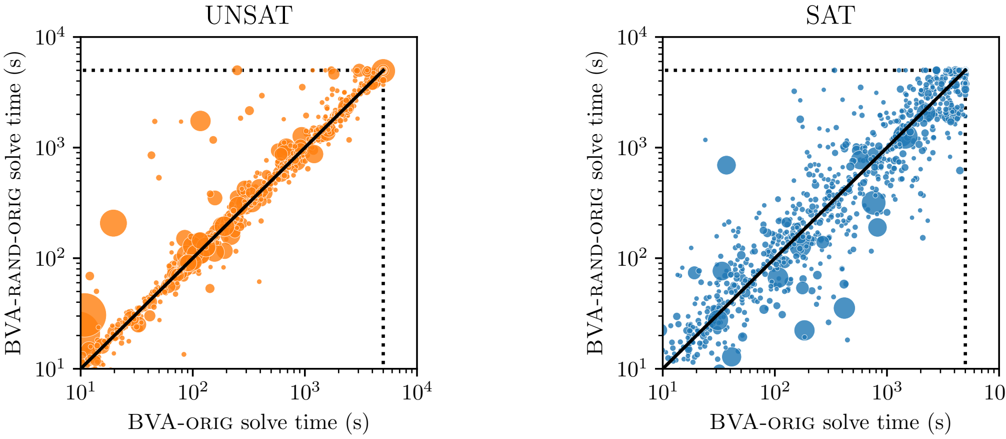
We address these questions directly in the following paragraphs:

A1: Formulas with larger compression factors tend to be solved faster, but this is not always the case. As demonstrated in Table 1, even small reductions from BVA can have a large impact on solve time. For example, on the packing k-coloring problem, a *single added variable* can reduce solve time by over a factor of 5 if picked correctly (Table 1).

We compute the *compression factor* of a formula as the ratio of the formula size *before* to the size *after* running BVA. For example, a factor of 1 indicates no reduction, a factor of 2 indicates the formula was reduced to 50% the original size, and a factor of 10 indicates the formula was reduced to 10% of the original size. Similarly, we compute the *speedup* as the ratio of solve time to BASELINE solve time (values below 1 indicate the formula was solved faster). In Figure 6 we plot the speedup of BVA-RAND-3HOP against the compression factor for every problem in the benchmark. Equivalent figures for BVA-ORIG and BVA-RAND-ORIG look similar and are available in the appendix (Figure A1 and Figure A2).

For formulas that could be greatly reduced, there is an observable trend towards a greater speedup. However, for small reductions, the speedup is much more variable. In some cases, even formulas that are reduced to less than 10% of the original size may be *slowed down* by BVA. With BVA-RAND-3HOP, 60% of formulas had a compression factor greater than 1, 40% had a factor greater than 2, and 4% had a factor greater than 10.

A2: Randomization is Detrimental to BVA. Randomization has a negative effect on the performance of BVA; in all benchmark groups, BVA-RAND-ORIG solved fewer formulas and has a higher PAR-2 score than BVA-ORIG. Interestingly, this effect appears to be entirely due to the *structure* of the resulting formula and not the resulting *size* of the formula.



(a) Relative solve time for UNSAT formulas.

(b) Relative solve time for SAT formulas.

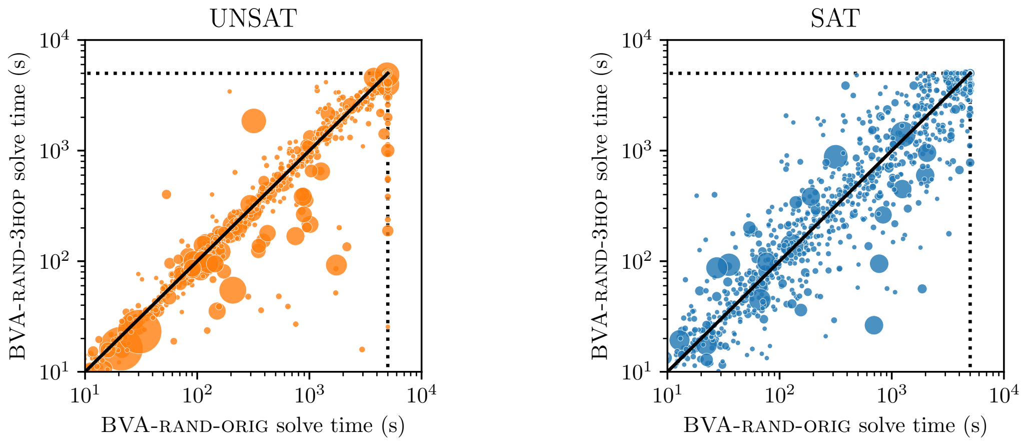
Figure 7 Difference in reduction size and solve time between BVA-RAND-ORIG and BVA-ORIG on formulas from ANNI-2022. Larger points indicate a more-reduced formula.

In Figure 7, we plot the relative solve times of formulas from the ANNI-2022 benchmark for BVA-RAND-ORIG and BVA-ORIG. While there is almost no difference in the reduction sizes of the formulas produced by BVA-RAND-ORIG and BVA-ORIG (formula sizes differ by less than 1.5%), a number of formulas were substantially slowed down (Figure 7). Note that in these plots, randomization prior to BVA is more detrimental for UNSAT formulas and introduces a lot of variance to SAT formulas.

A3: 3-Hop Heuristic is Robust to Randomization. While randomization has a negative effect on the original implementation of BVA, we observe that our heuristic-guided BVA is robust to this effect. Despite being provided with randomized formulas, it is able to generate high quality variable additions and recover all of the performance loss of BVA-RAND-ORIG, even surpassing BVA-ORIG in many cases on number of problems solved and PAR-2 score. We believe the slight performance improvement over BVA-ORIG in several cases is due to the presence of "pre-randomized" formulas in the benchmark; in these cases BVA-ORIG already suffers the effects of randomization while BVA-RAND-3HOP is able to recover the original structure of the problem.

In Figure 8, we compare the relative solve times of formulas from the ANNI-2022 benchmark for BVA-RAND-3HOP and BVA-RAND-ORIG. As in the previous section, the formula sizes between the two variants differs by less than 1.5% on average. However, BVA-RAND-3HOP is able to speed up many formulas, especially UNSAT instances.

A4: SBVA performs similar to BVA in most cases and performs extremely well for a few families. We found that both the original implementation of BVA and our heuristic-guided version have strong effects for specific families of formulas. In Figure 9, we plot the relative performance of the four configurations on 10 formula families for which BVA was effective. For these plots we allow BVA/SBVA to run for the full 5 000 seconds and consider only the CaDiCaL solve time in the plots in order to understand the effectiveness of the formula rather than the speed of BVA. In this section, we briefly describe some of the families where BVA was most effective.



(a) Relative solve time for UNSAT formulas.

(b) Relative solve time for SAT formulas.

Figure 8 Difference in reduction size and solve time between BVA-RAND-3HOP and BVA-RAND-ORIG on formulas from ANNI-2022. Larger points indicate a more-reduced formula.

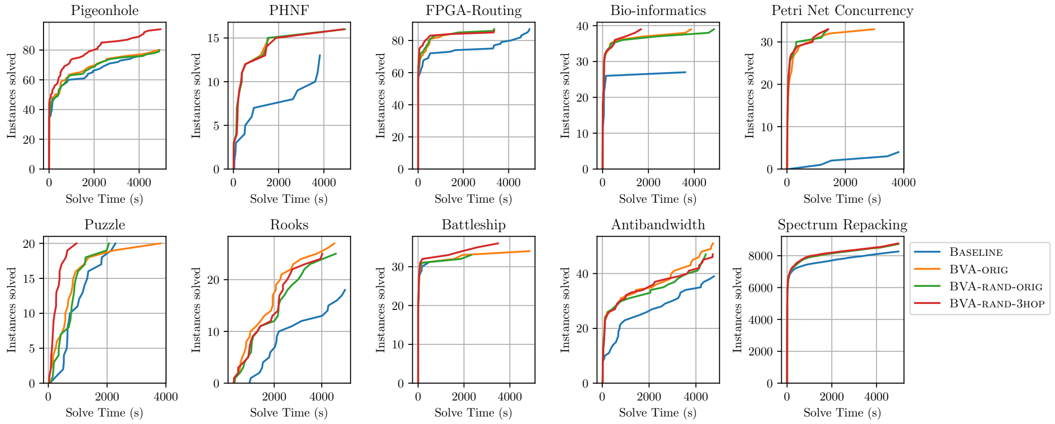


Figure 9 Performance of BVA/SBVA on 10 families of formulas where it was effective.

Pigeonhole / PHNF / FPGA-Routing. Pigeonhole formulas try to uniquely assign n pigeons to m holes. Like the packing k-coloring problem, these formulas consist primarily of AtLeastOne constraints (a pigeon must be in at least one hole) and pairwise AtMostOne constraints (two pigeons cannot share a hole). Our benchmark also contains variants of this problem, e.g. allowing multiple pigeons in a hole. These formulas are difficult for SAT solvers due to the number of possible permutations.

We found that SBVA was quite effective for UNSAT instances of pigeonhole problems (note that SAT instances of pigeonhole problems are trivial), able to solve new instances that the other three configurations could not solve. Interestingly, we found that these newly solved problems consist mainly of *pre-shuffled* pigeonhole problems. A full list of solved UNSAT pigeonhole problems is provided in Table A1. Other pigeonhole-like families in the dataset include PHNF (Pigeonhole Normal Form) [19] and FPGA-Routing [17], which consists of problems generated by combining two pigeonhole problems. All forms of BVA were very effective on these problems compared to BASELINE.

Petri Net Concurrency. Petri nets are a model of concurrent computation that consists of places and transitions [6]. They are used to model a variety of systems, including chemical reactions, manufacturing processes, and computer programs. The Petri Net Concurrency family consists of formulas that encode the satisfiability of Petri nets. All three configurations of BVA are able to generate very compact encodings for these formulas, with an average compression factor of more than 20.

Bioinformatics. The bioinformatics family consists of problems that encode genetic evolutionary tree computations into SAT [5]. As noted by the authors of the original BVA paper, these problems are also reduced significantly with BVA. For the problems in this family, we found that the average compression factor was more than 7 for all three BVA configurations, i.e. the formulas were reduced to less than 15% total size on average.

Puzzle / Rooks / Battleship. We found BVA to be useful in several families of formulas derived from 2-D games. The puzzle family consists of formulas that encode the satisfiability of a sliding-block puzzle and were contributed by van der Grinten to SAT Comp 2017. The rooks family asks if it is possible to place $N + 1$ rooks on a $N \times N$ chessboard such that no two rooks can attack each other [16]. The battleship family consists of problems that are derived from the battleship guessing game and were contributed by Skvortsov to SAT Comp 2011. BVA was effective in all three families and SBVA was especially effective for the puzzle and battleship families.

Antibandwidth / Spectrum Repacking. The antibandwidth [10] and spectrum repacking [18] formulas are both related to assigning radio stations to channels. Specifically, the antibandwidth family asks if it possible to assign a given set of stations to a given set of channels such that the difference in channel between any two stations is at least k. Similarly, the spectrum repacking family asks if it is possible to reassign a given set of stations into a smaller set of channels, taking into account physical distances between stations and the bandwidth of each channel. All configurations of BVA were effective on these problems.

7 Conclusion

Bounded Variable Addition is surprisingly effective at reducing the size of formulas and improving solve time by introducing auxiliary variables. We discovered that this speedup is caused not only by the reduction in formula size but also the introduction of certain *effective* auxiliary variables. We found that the original implementation was sensitive to randomization and proposed a new heuristic-guided implementation, SBVA, that is robust to this effect. In a competition-style benchmark, we show that using SBVA resulted in the most formulas solved in every category, outperforming both BVA and the baseline (no preprocessor). Additionally, SBVA was extremely effective on certain families of formulas, demonstrating that auxiliary variables can be useful in practice if they are chosen carefully.

References

1 Tomas Balyo, Marijn J.H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2022: Solver and Benchmark Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2022.

2 A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, NLD, 2009.

3 Armin Biere and Marijn Heule. The effect of scrambling CNFs. In *Proceedings of Pragmatics of SAT*, volume 59, pages 111–126, 2019.

4 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in SAT solving. *Handbook of Satisfiability*, 336:391–435, 2021.

5 Maria Luisa Bonet and Katherine St John. Efficiently calculating evolutionary tree measures using SAT. In *Theory and Applications of Satisfiability Testing-SAT 2009: 12th International Conference, SAT 2009, Swansea, UK, June 30-July 3, 2009. Proceedings 12*, pages 4–17. Springer, 2009.

6 Pierre Bouvier and Hubert Garavel. Efficient algorithms for three reachability problems in safe petri nets. In Didier Buchs and Josep Carmona, editors, *Application and Theory of Petri Nets and Concurrency*, pages 339–359, Cham, 2021. Springer International Publishing.

7 Shawn T. Brown, Paola Buitrago, Edward Hanna, Sergiu Sanielevici, Robin Scibek, and Nicholas A. Nystrom. Bridges-2: A platform for rapidly-evolving and data intensive research. In *Practice and Experience in Advanced Research Computing*, PEARC '21, New York, NY, USA, 2021. Association for Computing Machinery.

8 Stephen A Cook. A short proof of the pigeon hole principle using extended resolution. *Acm Sigact News*, 8(4):28–32, 1976.

9 Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. *SAT*, 3569:61–75, 2005.

10 Katalin Fazekas, Markus Sinnl, Armin Biere, and Sophie Parragh. Duplex encoding of staircase at-most-one constraints for the antibandwidth problem. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21–24, 2020, Proceedings 17*, pages 186–204. Springer, 2020.

11 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science.

12 Markus Iser and Carsten Sinz. A problem meta-data library for research in SAT. In Daniel Le Berre and Matti Järvisalo, editors, *Proceedings of Pragmatics of SAT 2015 and 2018*, volume 59 of *EPiC Series in Computing*, pages 144–152. EasyChair, 2019.

13 W. Klieber and G. Kwon. Efficient CNF encoding for selecting 1 from n objects. In *Fourth Workshop on Constraints in Formal Verification (CFV)*, 2007.

14 Petr Kučera, Petr Savickỳ, and Vojtěch Vorel. A lower bound on CNF encodings of the at-most-one constraint. *Theoretical Computer Science*, 762:51–73, 2019.

15 Norbert Manthey, Marijn JH Heule, and Armin Biere. Automated reencoding of boolean formulas. In *Haifa Verification Conference*, pages 102–117. Springer, 2012.

16 Norbert Manthey and Peter Steinke. Too many rooks. *Proceedings of SAT competition*, pages 97–98, 2014.

17 Gi-Joon Nam, Fadi Aloul, Karem Sakallah, and Rob Rutenbar. A comparative study of two boolean formulations of FPGA detailed routing constraints. In *Proceedings of the 2001 International Symposium on Physical Design*, pages 222–227, 2001.

18 Neil Newman, Alexandre Fréchette, and Kevin Leyton-Brown. Deep optimization for spectrum repacking. *Communications of the ACM*, 61(1):97–104, 2017.

19 O. Roussel. Another SAT to CSP conversion. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 558–565, 2004.

20 Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 827–831, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

21 Bernardo Subercaseaux and Marijn JH Heule. The packing chromatic number of the infinite square grid is at least 14. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

22 Bernardo Subercaseaux and Marijn JH Heule. The packing chromatic number of the infinite square grid is 15. *arXiv preprint*, 2023. arXiv:2301.09757.

23 Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Mathematics and Mathematical Logic 2*, pages 115–125, 1968.

A Appendix

A.1 Reduction Size vs. Solve Time

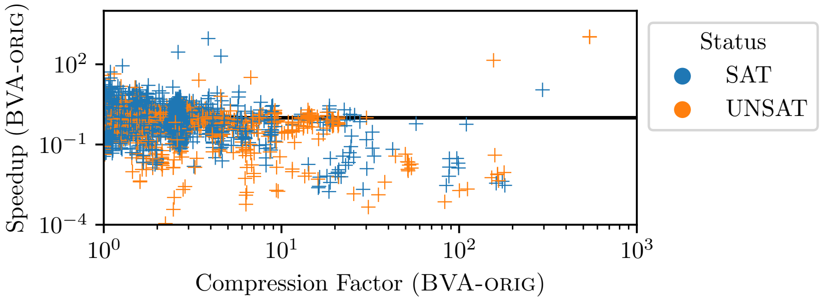


Figure A1 Formula speedup compared to compression factor for BVA-ORIG.

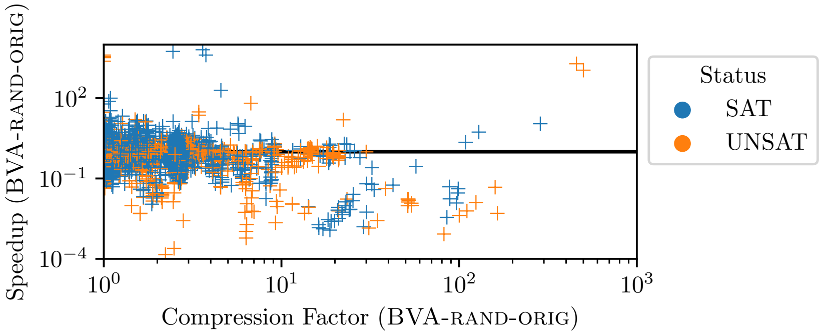


Figure A2 Formula speedup compared to compression factor for BVA-RAND-ORIG.

A.2 Performance on Pigeonhole Problems

Table A1 Performance on unsatisfiable instances of problems in the pigeon-hole family.

	Solve Time (s)			
Instance (unsatisfiable)	Baseline	BVA-orig	BVA-rand-orig	BVA-rand-3hop
a_rphp035_05	499.77	478.28	**327.31**	328.82
a_rphp045_05	2165.96	2069.67	**1748.25**	1952.39
a_rphp055_04	79.75	81.91	79.09	**75.39**
a_rphp065_04	168.98	164.66	**137.85**	146.82
a_rphp085_04	760.00	752.18	707.73	**661.66**
a_rphp098_04	**1945.51**	2136.84	2726.63	2318.10
ae_rphp035_05	410.54	501.24	426.94	**407.24**
ae_rphp045_05	2402.73	2549.29	**2206.80**	2307.20
ae_rphp055_04	83.73	86.48	81.64	**74.17**
ae_rphp075_04	513.63	515.97	558.53	**504.86**
ae_rphp095_04-sc2018	1937.08	1686.12	2074.11	**1678.99**
ae_rphp095_04	1706.39	1611.31	**1560.96**	1572.97
clqcolor-08-06-07.shuffled-as.sat05-1257	3.28	**0.62**	0.91	0.81
counting-clqcolor-unsat-set-b-clqcolor-08-06-07.sat05-1257.reshuffled-07	2.69	0.83	**0.76**	1.02
counting-easier-fphp-012-010.sat05-1214.reshuffled-07	111.76	33.26	30.03	**0.11**
counting-easier-fphp-014-012.sat05-1215.reshuffled-07	T.O.	T.O.	T.O.	**1.62**
counting-easier-php-012-010.sat05-1172.reshuffled-07	139.59	27.07	29.35	**3.45**
counting-easier-php-018-014.sat05-1175.reshuffled-07	T.O.	T.O.	T.O.	**4224.30**
counting-harder-php-014-013.sat05-1187.reshuffled-07	T.O.	T.O.	T.O.	**1760.18**
e_rphp035_05-sc2018	424.51	460.54	461.25	**387.80**
e_rphp035_05	482.43	480.30	**419.09**	501.65
e_rphp055_04	**73.62**	81.31	77.03	78.85
e_rphp065_04	139.17	**125.17**	143.14	144.34
e_rphp096_04	1626.72	1830.11	**1492.40**	1503.88
easier-fphp-020-015.sat05-1218.reshuffled-07	T.O.	T.O.	T.O.	**4205.14**
fphp-010-008.shuffled-as.sat05-1213	0.49	0.22	0.22	**0.02**
fphp-010-009.shuffled-as.sat05-1227	5.43	4.30	4.22	**0.06**
fphp-012-010.shuffled-as.sat05-1214	113.38	45.04	36.16	**0.12**
fphp-012-011.shuffled-as.sat05-1228	1722.86	1525.19	1844.10	**0.68**
fphp-014-012.shuffled-as.sat05-1215	T.O.	T.O.	T.O.	**1.95**
fphp-014-013.shuffled-as.sat05-1229	T.O.	T.O.	T.O.	**564.50**
fphp-016-013.shuffled-as.sat05-1216	T.O.	T.O.	T.O.	**383.94**
fphp-016-015.shuffled-as.sat05-1230	T.O.	T.O.	T.O.	**1027.46**
fphp-018-014.shuffled-as.sat05-1217	T.O.	T.O.	T.O.	**549.07**
fphp-020-015.shuffled-as.sat05-1218	T.O.	T.O.	T.O.	**944.65**
harder-fphp-016-015.sat05-1230.reshuffled-07	T.O.	T.O.	T.O.	**3496.64**
hole10.cnf.mis-98.debugged	2.20	**0.95**	1.51	1.05
ph9	5.68	1.55	4.02	**0.08**
ph10	120.63	13.83	50.17	**8.64**
ph11	3061.40	35.76	790.89	**26.45**
php-010-008.shuffled-as.sat05-1171	0.64	0.15	0.25	**0.03**
php-010-009.shuffled-as.sat05-1185	6.59	3.03	3.58	**0.07**
php-012-010.shuffled-as.sat05-1172	138.30	31.52	23.96	**3.02**
php-012-011.shuffled-as.sat05-1186	2695.99	1006.29	755.79	**26.97**
php-014-012.shuffled-as.sat05-1173	T.O.	T.O.	T.O.	**237.47**
php-016-013.shuffled-as.sat05-1174	T.O.	T.O.	T.O.	**3024.16**
php11e11	3599.98	**500.38**	780.09	796.55
rphp4_065_shuffled	146.66	148.59	**129.96**	132.01
rphp4_070_shuffled	**213.59**	249.18	280.29	227.72
rphp4_075_shuffled	448.36	459.23	**415.65**	419.72
rphp4_080_shuffled	532.00	519.81	516.74	**503.20**
rphp4_085_shuffled	666.98	723.74	654.52	**648.68**
rphp4_090_shuffled	853.68	**850.60**	919.62	890.40
rphp4_095_shuffled	1571.45	1362.84	1611.21	**1342.79**
rphp4_100_shuffled	2314.76	2545.95	2361.29	**2154.53**
rphp4_105_shuffled	3168.06	2948.21	2520.33	**2249.50**
rphp4_110_shuffled	3726.95	3389.56	**3208.10**	3665.63
rphp4_115_shuffled	4155.81	4406.08	4169.75	**4095.25**
rphp4_120_shuffled	T.O.	**4840.06**	4883.44	4948.49
rphp4_125_shuffled	T.O.	4613.90	4676.45	**3989.60**
rphp_p6_r28	T.O.	T.O.	**4895.39**	T.O.
tph6	226.78	12.06	68.97	**0.52**
tph7	T.O.	249.13	T.O.	**0.88**
tph8	T.O.	T.O.	T.O.	**188.30**

A.3 Performance on Bioinformatics Problems

Table A2 Performance on unsatisfiable instances of problems in the bioinformatics family.

	Solve Time (s)			
Instance (unsatisfiable)	Baseline	BVA-orig	BVA-rand-orig	BVA-rand-3hop
ndhf_xits_09_UNSAT	**T.O.**	**9.69**	11.10	10.52
ndhf_xits_10_UNSAT	**T.O.**	70.30	70.43	**63.04**
ndhf_xits_11_UNSAT	**T.O.**	612.65	869.25	**401.50**
ndhf_xits_12_UNSAT	**T.O.**	**T.O.**	**T.O.**	**1003.12**
rbcl_xits_06_UNSAT	5.27	0.20	0.22	**0.20**
rbcl_xits_07_UNSAT	110.14	0.60	0.60	**0.47**
rbcl_xits_08_UNSAT	3603.02	2.09	2.15	**1.72**
rbcl_xits_09_UNKNOWN	**T.O.**	**8.11**	9.55	10.69
rbcl_xits_10_UNKNOWN	**T.O.**	69.24	**56.82**	95.77
rbcl_xits_11_UNKNOWN-sc2009	**T.O.**	311.98	**309.87**	505.25
rbcl_xits_11_UNKNOWN	**T.O.**	**331.32**	392.19	528.49
rbcl_xits_12_UNKNOWN	**T.O.**	**3607.56**	4857.58	**T.O.**
rpoc_xits_07_UNSAT	54.86	**0.95**	1.03	0.97
rpoc_xits_09_UNSAT	**T.O.**	41.37	30.13	**24.29**
rpoc_xits_10_UNKNOWN	**T.O.**	237.25	**172.23**	182.65
rpoc_xits_11_UNKNOWN-sc2009	**T.O.**	3863.50	2369.91	**1672.29**
rpoc_xits_11_UNKNOWN	**T.O.**	1813.61	4624.68	**1413.96**

Table A3 Performance on satisfiable instances of problems in the bioinformatics family.

	Solve Time (s)			
Instance (unsatisfiable)	Baseline	BVA-orig	BVA-rand-orig	BVA-rand-3hop
ndhf_xits_19_UNKNOWN-sc2011	143.44	31.96	**9.59**	13.37
ndhf_xits_20_SAT	29.44	**2.20**	3.32	3.38
ndhf_xits_21_SAT	6.27	2.16	**1.13**	2.27
ndhf_xits_22_SAT	3.09	1.11	**0.50**	0.86
rbcl_xits_14_SAT	1.31	0.47	**0.46**	1.91
rbcl_xits_18_SAT	0.22	0.04	0.05	**0.03**
rpoc_xits_17_SAT	1.22	0.14	0.11	**0.11**

An Analysis of Core-Guided Maximum Satisfiability Solvers Using Linear Programming

George Katsirelos

Université Paris-Saclay, AgroParisTech, INRAE, UMR MIA Paris-Saclay, 91120, Palaiseau, France

Abstract

Many current complete MaxSAT algorithms fall into two categories: core-guided or implicit hitting set. The two kinds of algorithms seem to have complementary strengths in practice, so that each kind of solver is better able to handle different families of instances. This suggests that a hybrid might match and outperform either, but the techniques used seem incompatible. In this paper, we focus on PMRES and OLL, two core-guided algorithms based on max resolution and soft cardinality constraints, respectively. We show that these algorithms implicitly discover cores of the original formula, as has been previously shown for PM1. Moreover, we show that in some cases, including unweighted instances, they compute the optimum hitting set of these cores at each iteration. We also give compact integer linear programs for each which encode this hitting set problem. Importantly, their continuous relaxation has an optimum that matches the bound computed by the respective algorithms. This goes some way towards resolving the incompatibility of implicit hitting set and core-guided algorithms, since solvers based on the implicit hitting set algorithm typically solve the problem by encoding it as a linear program.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Combinatorial optimization; Theory of computation → Logic; Mathematics of computing → Solvers

Keywords and phrases maximum satisfiability, core-guided solvers, minimum hitting set problem, linear programming

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.12

Funding This work has been partly funded by the "Agence nationale de la Recherche" (ANR-19-PIA3-0004 ANITI-DIL chair of Thomas Schiex).

1 Introduction

MaxSAT is the optimization version of SAT, in which we are given a set of *hard* clauses which must always be satisfied, as well as a set of weighted soft clauses, with the objective to find an assignment which minimizes the weight of the falsified soft clauses. Much like the case for SAT, the performance of MaxSAT solvers has been steadily improving over the past few years [5]. Two classes of algorithms have contributed significantly to this improvement: implicit hitting set (IHS) solvers [12, 14, 13, 6, 8] and core-guided solvers [18, 2, 24, 23, 22, 19]. Both are based on iteratively calling a SAT solver on formulas derived from the original MaxSAT instance and extracting unsatisfiable cores, but they are very different in their operation. IHS solvers exploit the hitting set duality of cores and correction sets (solutions)[26], and they try to build up a collection of cores that are enough to make the minimum hitting set match the optimum solution. Crucially, IHS solvers only ask the SAT solver to extract cores from subsets of the initial MaxSAT instance, which are all approximately equally hard. Core-guided solvers, on the other hand, reformulate the input instance with each core they discover so that it exhibits a higher lower bound. The reformulation generates ever more constrained formulas, which get harder and harder.

Despite their different approaches, both classes of algorithms are competitive, but they perform well in different families of instances. Hence, it would be desirable to understand exactly how they relate to each other and build algorithms with the strength of both. In that

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 12; pp. 12:1–12:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

direction, Bacchus and Narodytska [7] showed that the cores discovered by the PM1 [18] algorithm correspond to a collection of cores of the original instance. Later, Narodytska and Bjørner [25] showed that for unweighted instances, PM1 actually discovers a hitting set of these cores of the original formula at every iteration. These results showed that there exists a close relationship between IHS and core-guided solvers.

Here, we focus on PMRES [24] and OLL [22]. Our contributions are as follows.

- We show that, like PM1, each core computed by PMRES and OLL corresponds to a set of cores of the original MaxSAT instance.
- We identify a condition for when the lower bound computed by PMRES or OLL matches the optimum hitting set of the set of cores of the original formula. This includes the case when the input instance is unweighted.
- We show that the hitting set problem over these cores can be formulated compactly as an integer linear program for both PMRES and OLL. Moreover, the linear relaxation of that ILP has a lower bound which is at least as great as the bound computed by PMRES or OLL, respectively.
- The linear program that we give is actually a subset of a higher level relaxation of that hitting set problem in the Sherali-Adams hierarchy [28].

The first two contributions match what has been done for PM1 previously, although our proofs are notably simpler, owing to the fact that the cores of PMRES and OLL have a much more regular structure than those of PM1. The latter two contributions provide further insight into the relationship between these core-guided algorithms and IHS. The LP formulation points the way to an algorithm that combines features of both core-guided and implicit hitting set solvers, since IHS solvers typically solve the hitting set problem with an ILP solver: any bounds computed by PMRES or OLL can be imported into IHS by way of this LP. The fact that this LP is a subset of a high level Sherali-Adams relaxation also shows IHS and core-guided solvers as being two extreme instantiations of the same algorithmic framework, where both solvers try to solve an implicit hitting set problem. But whereas IHS discovers only cores of the original formula and offloads solving of the hitting set problem to an external solver, PMRES very aggressively searches for a non-obvious set of new variables to add to the linear relaxation of the hitting set problem, in order to keep it as close as possible to the optimum integer solution, but places a great burden on the SAT solver. This suggests a more effective tradeoff could be found.

2 Background

In addition to the basics of MaxSAT, we also introduce necessary background on linear programming and weighted constraint satisfaction problems (WCSPs).

2.1 Satisfiability

A SAT formula ϕ in conjunctive normal form (CNF) is a conjunction of clauses and a clause is a disjunction of literals. We also view a CNF formula as a set of clauses and a clause as a set of literals. For a CNF formula F, we write $vars(F)$ for the set of all variables whose literals appear in the clauses of F. The Weighted Partial MaxSAT (WPMS) problem is a generalization of SAT to optimization. A WPMS formula is a triple $W = \langle H, S, w \rangle$ where H is a set of *hard* clauses, S is a set of *soft* clauses and $w : S \to \mathbb{R}_{\geq 0}$ is a cost function over the soft clauses. We also write $H(W) = H, S(W) = S, vars(W) = vars(H) \cup vars(S)$. For an assignment I over $vars(W)$, we overload notation to write $w(I) \triangleq \sum_{c \in S : I \vdash \neg c} w(c)$ for the cost of the soft clauses that I falsifies. The objective is to find an assignment I to $vars(W)$ such that all clauses in H are satisfied and the cost of the falsified soft clauses, i.e., $w(I)$, is minimized. We write $opt(W) \triangleq min_I w(I)$ for this value. A WPMS formula $\langle H, S, w\rangle$ with $w(c) = 1$ for all $c \in S$, is a partial MaxSAT formula. If, additionally, H is empty, it is a MaxSAT formula.

Two WPMS formula $W = \{H, S, w\}$ and $W' = \{H', S', w'\}$ are *equivalent* if for each assignment I to $vars(W)$ that satisfies H, we can extend it to an assignment I' to $vars(W')$ that satisfies H' and $w(I) = w'(I') + b$, for some constant b that is the same for all assignments. For example, $W = \{\emptyset, \{(x), (\overline{x})\}, w\}$, where $w((x)) = 5, w((\overline{x})) = 3$ is equivalent to $W' = \{\emptyset, \{(x)\}, w'\}$, where $w'((x)) = 2$, because the weight of all assignments differs by 3 in W, W'. This notion of equivalence is important in our subsequent analysis.

Given an unsatisfiable CNF formula F, a set $C \subset F$ is a core of F if C is unsatisfiable. If C is minimal by set inclusion, it is a MUS (minimal unsatisfiable subset) of F. Given a WPMS formula $W = \langle H, S, w\rangle$, a set $C \subseteq S$ is a core of W if $H \cup C$ is unsatisfiable. In the rest of this paper, we assume for simplicity that H is satisfiable and $H \cup S$ is unsatisfiable.

In the sequel, we make some assumptions without loss of generality. First, we assume that all soft clauses in a MaxSAT formula $W = \langle H, S, w\rangle$ are unit. If there exists a clause $c_i \in S$ which is not unit, we create the formula $W' = \langle H', S', w'\rangle$ with $H' = H \cup \text{cnf}(\neg c_i \iff b_i)$, $S' = S \cup \{(\overline{b}_i)\} \setminus \{c_i\}$, where b_i is a fresh variable, called the *blocking variable* for c_i, and $w'((\overline{b}_i)) = w(c_i)$, $w'(c) = w(c)$ for all $c \in S \cap S'$. We see that W is equivalent to W by noting that we can extend any assignment of W to W' by setting b_i so that it satisfies $b_i \iff \neg c_i$. Moreover, we assume that the unique literal in all soft clauses appears with negative polarity. If this does not hold, we can make it so by renaming. Because of this assumption, we identify each soft clause with the unique variable it contains and we use that literal to refer to it. Finally, we assume that there exist no soft clauses with cost 0, as we can remove those without affecting satisfiability or cost. However, we use the convention that $w(x) = 0$ for all positive literals and all negative literals of variables that do not appear in a soft clause. Given this convention, a WPMS instance can be written as $W = \langle H, w\rangle$, and S is implicitly $S = \{(\overline{x}_i \mid w(\overline{x}_i) > 0\}$. We use the two formulations interchangeably.

Solving WPMS

Most current SAT solvers have the ability to not only report SAT or UNSAT for a given formula, but also, given a partition of its clauses so that $\phi = \psi \cup \chi$, report a subset of χ such that $\psi \cup \chi$ is unsatisfiable. In terms of WPMS, it means a modern SAT solver can give a subset of S such that $H \cup S$ is unsatisfiable, i.e., a core of the WPMS formula. Because we have assumed that S contains negative unit clauses only, it follows that each core of W is a positive clause entailed by H.

The *implicit hitting set* (IHS) algorithm for WPMS [12, 14, 13, 6, 8] is based on the observation that the set of soft clauses $CS \subseteq S$ violated by a solution I is a hitting set of the set of all cores of W [26]. Hence, an optimal solution is a minimum hitting set of the cores of W. Hitting sets of all cores are called correction sets.

The IHS algorithm maintains an initially empty set of discovered cores $\mathcal{C}$ of W and a minimum hitting set of $\mathcal{C}$, $hs(\mathcal{C})$. If the SAT formula $H \cup (S \setminus hs(\mathcal{C}))$ is satisfiable, then its solutions are optimal solutions of W and $w(hs(\mathcal{C})) = w(W)$. Otherwise, a new core is extracted and added to $\mathcal{C}$ and the loop repeats. Actual implementations of the IHS algorithm in MaxHS [12] and LMHS [27, 9] contain many optimizations over this basic loop.

A *core-guided* algorithm for WPMS [18, 24, 23, 22, 19] is an iterative algorithm that generates a sequence of WPMS instances $W^0 = \langle H^0, w^0\rangle = W, \ldots, W^m = \langle H^m, w^m\rangle$ and a sequence of lower bounds $lb^0 = 0 < lb^1 < \ldots < lb^m$ such that $H^i \models H^{i-1}$ for all $i \in [1, m]$

and W^0 is equivalent to W^i for all $i \in [1, m]$ and the weights of the assignments differ by lb_i, therefore $opt(W) = lb_i + opt(W^i)$. Moreover, in the last iteration it holds $opt(W^m) = 0$, so $opt(W) = lb^m$. In words, a core-guided algorithm generates a sequence of equivalent WPMS instances such that each successive instance is used to derive an increased lower bound for the original instance, while decreasing the cost of every solution by the same amount. The final instance admits a solution with zero weight, and each such solution of W^m is an optimal solution of W. All such solutions are solutions of the SAT formula $H^m \mid_0$, defined as $H^m \cup (\overline{x}) \mid w(x) > 0$, i.e., with all soft clauses made into hard clauses. In order to derive each successive instance W^{i+1} in the sequence, it extracts a core from W^i and uses it to transform it into W^{i+1} and increase the lower bound, hence the name core-guided. The algorithms we study here, PMRES and OLL, are core-guided algorithms. Following Narodytska and Bjørner [25], we call cores of W^i for $i > 0$ *meta cores*, or metas, to distinguish them from cores of the original formula W^0. We write m^i for the meta discovered at iteration i.

2.2 Linear programming and Weighted Constraint Satisfaction

An integer linear program (ILP) IP has the form $\min c^T x : Ax \geq b \wedge x \in \mathbb{Z}_{\geq 0}$, where x is a vector of n variables, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$. For a given x, if $Ax \geq b$, then it is a feasible solution of IP. We write $c(x) = c^T x$ for the *cost*[1] of x. We write $c(IP)$ for the cost of a feasible solution with minimum cost. The *linear relaxation* P of IP is the problem $\min c^T x : Ax \geq b \wedge x \in \mathbb{R}^n_{\geq 0}$, i.e., one where we relax the integrality constraint $x \in \mathbb{Z}^n_{\geq 0}$. This is called a linear program (LP). Linear programs have the *strong duality* property, namely that for every linear program P in the above form, there exists another linear program $P^D = \max b^T y : A^T y \leq c \wedge y \in \mathbb{R}^m_{\geq 0}$, with the property that $c_{P^D}(\hat{y}) \leq c_P(\hat{x})$ for every feasible solution $\hat{x}$ of P and $\hat{y}$ of P^D and $c_{P^D}(y^*) = c_P(x^*)$ for optimal solutions x^* and y^*. Given a feasible dual solution $\hat{y}$, the value $A_i^T y - c_i$, the slack of the dual constraint corresponding to the primal variable x_i, is called the *reduced cost* of x_i, denoted $rc_i(\hat{y})$. A necessary condition for optimality called *complementary slackness* links the two solutions: $x_i^* rc_i(y^*) = 0$, i.e., for each variable x_i, either it is zero or its corresponding dual constraint $(A_i y \leq c_i)$ is tight (has *zero slack*).

A Boolean Cost Function Network (CFN) is a pair $\langle V, D, C\rangle$, where V is a set of variables, D is a function mapping variables to domains, and C is a set of cost functions. If the domain of a variable v is binary, we write v for the value $v = 1$ and $\overline{v}$ for $v = 0$. Each cost function is a pair $\langle S, c\rangle$ where $S \subseteq V$ is its scope and c_S is a function $\prod_{x \in S} D(x) \to \mathbb{R}_{\geq 0} \cup \infty$. We assume there exists at most one cost function for each scope, so c_S is a shortcut for $\langle S, c_S\rangle$. An assignment I_S to a scope S is a function which maps every variable $x \in S$ to a value in $D(x)$. When we omit S, it means $S = V$. When convenient, we also use I to denote the set $\{v = a \mid I(v) = a, v \in V\} \cup \{v \neq b \mid I(v) \neq b, v \in V, b \in D(x)\}$. For a scope S and assignment I, $I_{\downarrow S}$ is the projection of I to S. $t(S)$ denotes all possible assignments to S.

We use the convention that for a cost function c_S, $c_S(I) = c_S(I_{\downarrow S})$, i.e., we implicitly project to S. For a CFN P, we write $c_P(I) = \sum_{c_S \in F} c_S(I)$. The Weighted Constraint Satisfaction Problem (WCSP) is to find an assignment I such that $c_P(I) < \infty$ and that minimizes c_P. The term WCSP is often used to refer both to the underlying CFN and to the optimization problem, and we do the same here. Additionally, we assume the existence of a unary cost function $c_{\{v\}}$ (abbreviated as c_v) for every variable $v \in V$ and a nullary cost function $c_\varnothing$, which is a lower bound for c_P, becayse all costs are non-negative. A CSP is a WCSP in which the domain of all cost functions is $\{0, \infty\}$.

[1] We stick to the terminology of *weights* in MaxSAT and *costs* in ILP and WCSP, even though they serve the same purpose.

A WCSP $P = \langle V, C\rangle$ can be formulated as the following ILP:

$$\begin{aligned}
&\min \sum_{c_S \in C, l \in t(S)} c_S(l) x_{Sl} && && (1)\\
&\text{s.t.} && && (2)\\
&x_{\{v\},a} = \sum_{l \in t(S): v=a \in l} x_{Sl} && \forall v \in V, a \in D(v), c_S \in C && (3)\\
&\sum_{a \in D(v)} x_{\{v\},a} = 1 && \forall v \in V && (4)\\
&x_{Sl} \in \mathbb{Z}_{\geq 0} && \forall v \in V, c_S \in C, l \in t(S) && (5)
\end{aligned}$$

The linear relaxation of (1)– (5) defines the *local polytope* of P. A dual feasible solution of the local polytope LP has a particular interpretation: it defines a *reformulation* of the WCSP. A reformulation can be seen as a set of operations on a WCSP P that create a new WCSP $\hat{P}$ with modified costs, but $c_P(I) = c_{\hat{P}}(I)$ for all I. Therefore, a reformulation is said to *preserve equivalence*. This notion of equivalence is identical to the equivalence preserved by core-guided algorithms, with the primary difference being that the lower bound is explicitly represented in a WCSP in $c_\varnothing$. These operations can intuitively be thought of as moving cost among cost functions:

- Extention: $ext(v = a, c_S, \alpha)$, with $v \in S, a \in D(v)$. This subtracts cost α from $c(\{v\}, a)$ and adds it to $c(S, l)$ for all tuples $l \in t(S) : (v = a) \in l$. To see the correctness of this, consider the subset of the objective function $c_v(a)x_{\{v\},a} + \sum_{l \in t(S):(v=a)\in l} c_S(l)x_{Sl}$, as well as constraint (3). Since $x_{\{v\},a}$ is equal to the sum, the value of the objective remains unchanged by adding α to one and subtracting it from the other.
- Projection: $prj(c_S, v = a, \alpha)$, with $v \in S, a \in D(v)$. This is the same as $ext(v, c_S, -\alpha)$.
- Nullary projection: $prj_0(c_S, w)$. This subtracts cost α from each tuple $l \in t(S)$ and moves it to $c_\varnothing$. This is justified because $\sum_{l \in t(S)} x_{Sl} = 1$ and the cost of $c_\varnothing$ is a constant in the objective function.

Because these operations preserve equivalence, they are called Equivalence Preserving Transformations (EPTs). A valid set of EPTs ensures that all cost functions are non-negative everywhere, but there are valid sets of EPTs for which any sequence of performing them leaves intermediate negative costs. A valid set of EPTs can be mapped to a feasible dual solution of the local polytope LP and vice versa. A set of EPTs which achieves the greatest increase in $c_\varnothing$, and hence the lower bound, can be mapped to an optimal dual solution of the local polytope LP [11]. Given a dual solution, the cost of each tuple $l \in t(S)$ is given by the reduced cost of the variable x_{Sl}.

For a WCSP P, let $Bool(P)$ be the CSP (not weighted) defined by accepting exactly those tuples which have cost 0, i.e., changing all costs which are greater than 0 to ∞. Let $\hat{P}$ be a reformulation of P. A consequence of complementary slackness is that if $\hat{P}$ is an optimal reformulation, then $Bool(\hat{P})$ has a non-empty arc consistency closure [11, 15], in which case it is said that $\hat{P}$ is virtually arc consistent (VAC). This is not a sufficient condition for optimality, however. Conversely, if $\hat{P}$ is not VAC, therefore $Bool(\hat{P})$ has an empty arc consistency closure, there exists a reformulation with a higher $c_\varnothing$.

3 PMRES

The PMRES algorithm is a core guided solver which was introduced by Narodytska and Bacchus [24] and is implemented primarily in the Eva solver. We describe it briefly here. In this description, we use the view of WPMS as hard and soft clauses, rather than hard clauses and an objective, because the transformations performed by PMRES temporarily violate the assumptions that allow us to take this alternative view. However, these assumptions are always restored at the end of each iteration.

3.1 Max-Resolution

Max resolution [20] is a complete inference rule for MaxSAT [10]. It consists of the following rule on soft clauses, in which the conclusions replace the premises:

$$\begin{array}{c} (A \lor x, w) \\ (B \lor \overline{x}, w) \\ \hline (A \lor B, w) \\ (A \lor x \lor \overline{B}, w) \\ (B \lor \overline{x} \lor \overline{A}, w) \end{array}$$

The first clause in the conclusions is equivalent to what resolution derives. The latter two are called compensation clauses, as they compensate for the cost of assignments which do not falsify the conclusion $A \lor B$ but falsify one of the discarded premises. Depending on the exact form of A and B, the compensation "clauses" may not actually be in clausal form and would have to be converted to a set of clauses each. We ignore this complication here, as our presentation of PMRES mostly avoids this case.

The rule is generalized to clauses with different costs $w_1 > w_2$ by cloning the heavier clause into clauses with costs w_2 and $w_1 - w_2$. When one of the clauses is hard, e.g., $w_1 = \infty$, we keep it in the conclusions.

Max resolution has the property that if W and $\hat{W}$ are the formulas before and after application of the rule, then they are equivalent.

3.2 Max-Resolution with cores

PMRES uses the specialization of this rule for a binary clause and a unit clause, i.e., $|A| = 1, B = \emptyset$.

$$\begin{array}{c} (A \lor x, w) \\ (\overline{x}, w) \\ \hline (A, w) \\ (\overline{x} \lor \overline{A}, w) \end{array}$$

As a core-guided solver, PMRES is an iterative algorithm and the first step in each iteration is to extract a meta core from W^i, or terminate if $H^i \cup S^i$ is satisfiable. Suppose that the meta is $m^i = \{b^i_1, b^i_2, \ldots, b^i_{r^i}\} \subseteq S^{i-1}$ and $w^i_{\text{min}} = \min_{b_j \in C} c^i(b^i_j)$. This implies the presence of the soft clauses $(\overline{b}^i_1, w_1), \ldots, (\overline{b}^i_{r^i}, w_{r^i})$. PMRES first splits each soft clause $(\overline{b}^i_j, w')$ with $w' > w^i_{\text{min}}$ into $(\overline{b}^i_j, w^i_{\text{min}})$ and $(\overline{b}^i_j, w' - w^i_{\text{min}})$. This temporarily violates our assumption that each soft clause contains a unique literal, but as we will see, this invariant is restored before the next iteration starts. In the next step, it adds to H^{i+1} the hard clause corresponding to C using the CNF encoding of $(b^i_1 \lor d^i_1), (d^i_1 \iff b^i_2 \lor d^i_2), \ldots (d^i_{r^i-2} \iff$

$(b_1 \vee b_2 \vee b_3 \vee b_4)$
$(b_5 \vee b_2)$
$(b_5 \vee b_3 \vee b_4)$

Figure 1 Cores of the instance used in the running example.

$b^i_{r^i-1} \vee d^i_{r^i_1}), (d^i_{r^i-1} \iff b^i_r)$, where $d^i_1, \ldots, d^i_{r-1}$ are fresh variables. It is clear that we can recover the clause $(b^i_1 \vee \ldots \vee b^i_r)$ by resolving (not with max-resolution, as the clauses are all hard) the first two clauses on d^i_1, then on d^i_2, and so on, therefore the encoding and the clause are equivalent. PMRES then applies max-resolution as follows:

Premises		Conclusions
$(b^i_1 \vee d_1, w^i_{\min})$	$(\overline{b}^i_1, w^i_{\min})$	$(d_1, w^i_{\min}), \boxed{(\overline{b}^i_1 \vee \overline{d}_1, w^i_{\min})}$
$(d_1, w^i_{\min})$	$(\overline{d}_1 \vee b^i_2 \vee d_2, w^i_{\min})$	$(b^i_2 \vee d_2, w^i_{\min}), ((\overline{b^i_2 \vee d_2}) \vee d_1, w^i_{\min})$
$\vdots$		
$(b^i_{r-1} \vee d_{r-1}, w^i_{\min})$	$(\overline{b}^i_{r-1}, w^i_{\min})$	$(d_{r-1}, w^i_{\min}), \boxed{(\overline{b}^i_{r-1} \vee \overline{d}_{r-1}, w^i_{\min})}$
$(d_{r-1}, w^i_{\min})$	$(\overline{d}_{r-1} \vee b^i_r, w^i_{\min})$	$(b^i_r, w^i_{\min}), (\overline{b}^i_r \vee d_{r-1}, w^i_{\min})$
$(b^i_r, w^i_{\min})$	$(\overline{b}^i_r, w^i_{\min})$	$\boxed{(\square, w^i_{\min})}$

The non-clausal constraints in light gray are tautologies and can be discarded. For example, by $(d^i_1 \iff b^i_2 \vee d^i_2)$, $(\overline{b^i_2 \vee d^i_2}) \vee d^i_1$ is equivalent to $(\overline{d}^i_1 \vee d^i_1)$, a tautology. The clauses in gray are used as input for the next max-resolution step. The framed clauses are new soft clauses that are kept for the next iteration. Since they are not unary, they are reified using fresh variables and converted to unit soft clauses, e.g., $f \iff b^i_1 \wedge d^i_1$ and $(\overline{f}, w)$, where f is the fresh variable. Finally, the empty soft clause $(\square, w_{\min})$ is used to increase the lower bound for the next iteration by $w_{\min}$.

Consider now a clause $(\overline{b}^i_j, w')$ that was split into two clones $(\overline{b}^i_j, w_{\min})$ and $(\overline{b}^i_j, w' - w_{\min})$. The former is consumed by max-resolution, therefore the invariant that each soft clause contains a unique literal is restored. This also allows us to implement the cloning process as a simple update: $w^{i+1}(b_j) = w^i(b_j) - w_{\min} = w' - w_{\min}$. If it happens that $w' = w_{\min}$, we maintain by the previously mentioned convention that $w^{i+1}(b_j) = 0$.

In the following, we write H^i_R for the formula consisting only of the clauses introduced by PMRES, therefore $H^i = H \cup H^i_R$. We also write F^i and D^i for the set of all variables, introduced to reify soft clauses (e.g. f above) or to encode the meta core clause (the d^i_j variables above), respectively. It has also been previously noted [25, 3] that the conjunction of the definitions of the F and D and the clauses $(b^i_1 \vee d^i_1)$ define a monotone circuit, with a binary gate corresponding to each $v \in F^i \cup D^i$, an unnamed $\vee$ gate corresponding to the clause $(b^i_1 \vee d^i_1)$, and an implicit $\wedge$ gate whose inputs are the unnamed $\vee$ gates, which is the output of the circuit.

▶ **Example 1** (Running Example). Consider an instance W with 5 soft clauses with cost 1 each and corresponding literals $b_1, \ldots, b_5$, and the cores shown in Figure 1. We show a run of PMRES in Figure 2 (for readability, we show the objective function rather than the set of soft clauses) that discovers first the core $(b_1 \vee b_2 \vee b_3 \vee b_4)$. It increases the lower bound by 1, adds the variables $D^1 = \{d^1_1, d^1_2, d^1_3\}$ and $F^1 = \{f^1_1, f^1_2, f^1_3\}$, defined as shown in the row corresponding to iteration 1. Since weights are unit, all original variables except b_5 disappear from the objective. In the next iteration, PMRES discovers the meta $\{b_5, f^1_2\}$, increases the

Iteration	Meta	New clauses	Objective
1	$\{b_1, b_2, b_3, b_4\}$	$d_1^1 \iff b_2 \vee d_2^1$, $d_2^1 \iff b_3 \vee d_3^1$,	
		$d_3^1 \iff b_4$,	
		$f_1^1 \iff b_1 \wedge d_1^1$, $f_2^1 \iff b_2 \wedge d_2^1$,	
		$f_3^1 \iff b_3 \wedge d_3^1$	$1 + b_5 + f_1^1 + f_2^1 + f_3^1$
2	$\{f_2^1, b_5\}$	$d_1^2 \iff b_5$	
		$f_1^2 \iff b_7 \wedge d_1^2$	$2 + f_1^1 + f_3^1 + f_1^2$

Figure 2 PMRES on the running example.

lower bound to 2, and introduces the variables d_1^2 and f_1^2. In the next iteration, the instance is satisfiable. One of the possible solutions is b_4, b_5, with cost 2, which matches the lower bound.

3.3 Cores and Hitting Sets of PMRES

We first observe that the f^i and d^i variables created on iteration i are functionally dependent on the b^i variables. Therefore, the formula H^i generated after the ith iteration is logically equivalent to H, i.e., every solution of H can be extended to exactly one solution of H^i.

▶ **Lemma 2.** *There exists a set $\mathcal{C}^i$ such that m^i is a core of H^i if and only if for each $c \in \mathcal{C}^i$, c is a core of ϕ.*

Proof. The set $\mathcal{C}^i$ can be derived from m^i and H_R^i by forgetting the variables f and d that were introduced by PMRES. More concretely, let $E^0 = \{m^i\}$. If there exists $c \in E^j$ such that $f \in c$ and f was introduced by PMRES and defined as $f \iff b \wedge d$, we set $E^{j+1} = E^j \setminus \{c\} \cup \{c \setminus \{f\} \cup \{b\}, c \setminus \{f\} \cup \{d\}\}$, i.e., we replace c by two clauses which have b and d, respectively, instead of f. If there exists $c \in E^j$ such that $d \in c$ and d was introduced by PMRES and defined as $d \iff b \vee d'$, we set $E^{j+1} = E^j \setminus \{c\} \cup \{c \setminus \{d\} \cup \{b, d'\}\}$, i.e., we replace d by b, d' in c. The process eventually terminates because it removes one reference to a variable introduced by PMRES and replaces it by a variable corresponding to a gate at a deeper level of the Boolean circuit defined by H_R^i, hence all variables must eventually be original variables of W^0. It is also confluent because the choice of variable to forget does not hinder other choices.

Since both forgetting variables and introducing functionally defined variables are satisfiability-preserving operations, we have $m^i \wedge H_R^i \models \mathcal{C}^i$ and $\mathcal{C}^i \models m^i \wedge H_R^i$. ◀

▶ **Lemma 3.** *Let $hs \subseteq S$. Then hs as an assignment can be extended to a solution of H_R^i if and only it is a hitting set of $\mathcal{C}_\cup^i$.*

Proof. This follows from lemma 2.

($\Rightarrow$) hs satisfies H_R^i, hence it satisfies all clauses in $\mathcal{C}^i$, which are cores, so it hits all the cores.

($\Leftarrow$) hs is a hitting set of $\mathcal{C}^i$, hence it satisfies all the corresponding clauses, hence it satisfies H_R^i. ◀

In the following, let $\mathcal{C}_\cup^i = \cup_{j \in [1,i]} \mathcal{C}^j$.

▶ **Observation 4.** *$\langle H_R^i, w^0 \rangle$ and $\langle H_R^i, w^i \rangle$ are equivalent.*

Proof. Consider $H_0 = \langle H_R^i, w^0 \rangle$. We know that m^0 is a core of H_0. By applying max resolution to m^0 as described in section 3.2, we get new variables and soft clauses. But these new variables are defined identically to the variables PMRES introduced to get H_R^1, which is a subset of H_R^i. Hence, we can identify them. By correctness of PMRES, we get that $\langle H_R^i, w^1 \rangle$ is equivalent to $\langle H_R^i, w^0 \rangle$. We apply the same argument inductively to complete the proof. ◀

▶ **Corollary 5.** *The WPMS $W_i^{hs} = \langle H_R^i, w^i \rangle$ encodes the minimum hitting set problem over $\mathcal{C}_\cup^i$, with weights shifted by lb^i. Hitting sets with cost lb^i, if they exist, are solutions of W_{hs}^i that use only soft clauses with soft 0.*

Proof. From Lemma 3, $\langle H_R^i, w^0 \rangle$ encodes minimum hitting set over $\mathcal{C}_\cup^i$. From Observation 4, $\langle H_R^i, w^0 \rangle$ and $\langle H_R^i, w^i \rangle$ are equivalent, therefore W_i^{hs} encodes minimum hitting set over $\mathcal{C}_\cup^i$.

The second part follows from the fact that, for any assignment I, $w^0(I) = lb_i + w^i(I)$, so if $w^0(I) = lb_i$, then $w^i(I) = 0$. ◀

Let us denote by $H_R^i \mid_0$ the formula H_R^i with all variables x such that $w(x) > 0$ set to false so that all models of $H_R^i \mid_0$ are minimum hitting sets of $\mathcal{C}^i$. Therefore if $H_R^i \mid_0$ is satisfiable, the bound computed by PMRES matches the cost of the minimum hitting set of $\mathcal{C}_\cup^i$.

▶ **Lemma 6.** *If W is a PMS instance, $H_R^i \mid_0$ is satisfiable for all iterations i of* PMRES.

Proof.

- All variables in D have cost 0.
- Moreover, all variables which appear in any meta have cost 0, because it is moved away by max-resolution.
- Therefore, all variables in $b_1^j, \ldots, b_{r^j}^j$ for $j \in [1, i]$ have zero cost.

We construct a solution to $H_R^i \mid_0$ by setting to false all variables which are inputs to false $\wedge$-gates (which is done by unit propagation), then we set variables to true by traversing metas in reverse chronological order:

1. For m^i, we pick the first variable in $b_1^j, \ldots, b_{r^j}^j$ and set it to true. We set all variables in F^i and D^i to false (the former is required for m^i because, as the last discovered core, all variables in F^i have non-zero weight.
2. Supposing we have satisfied all metas $m^{j+1}, \ldots, m^i$, consider m^j. Suppose that $0 \le q < |m^j|$ variables in F^j that have been set to true by previous steps, with indices $P^j = \{p_1, \ldots, p_q^j\}$. For simplicity of notation, assume that if P^j is empty, then $p_q^j = 0$. Then we set to true the variables $b_r^j \mid r \in P^j$ as well as $b_{p_q+1}^j$, and set the rest to false. When $p_q^j = 0$, this reduces to setting the first variable in b_1^j to true.
   a. This assignment satisfies the constraints introduced in H_R^j.
   b. Moreover, all the variables that appear in m^j have cost 0 after the j^{th} iteration. Therefore they cannot appear in any meta discovered in iterations $j+1, \ldots, i$ and the assignment we have chosen here does not contradict the assignments chosen in iterations $j+1, \ldots, i$. ◀

We can see where the proof of Lemma 6 breaks when applied to WPMS: the assertion 2b does not hold, because a variable whose cost has not been reduced to 0 may appear in later metas and our procedure may therefore create a conflicting assignment.

Iteration	Core	New clauses	Objective
1	$\{b_1, b_2, b_3, b_4\}$	$d_1^1 \iff b_2 \vee d_2^1$, $d_2^1 \iff b_3 \vee d_3^1$, $d_3^1 \iff b_4$, $f_1^1 \iff b_1 \wedge d_1^1$, $f_2^1 \iff b_2 \wedge d_2^1$, $f_3^1 \iff b_3 \wedge d_3^1$	$1 + b_2 + 2b_3 + 3b_4 + 5b_5 +$ $f_1^1 + f_2^1 + f_3^1$
2	$\{f_2^1, b_5\}$	$d_1^2 \iff b_5$ $f_1^2 \iff f_2^1 \wedge d_1^2$	$2 + b_2 + 2b_3 + 3b_4 + 4b_5 +$ $f_1^1 + f_3^1 + f_1^2$
3	$\{b_3, b_4, b_5\}$	$d_1^3 \iff b_4 \vee d_2^3$, $d_2^3 \iff b_5$ $f_1^3 \iff b_3 \wedge d_1^3$, $f_2^3 \iff b_4 \wedge d_2^3$	$4 + b_2 + b_4 + 2b_5 +$ $f_1^1 + f_3^1 + f_1^2 + 2f_1^3 + 2f_2^3$
4	$\{b_2, b_5\}$	$d_1^4 \iff b_5$ $f_1^4 \iff b_2 \wedge d_1^4$	$5 + b_4 + b_5 +$ $f_1^1 + f_3^1 + f_1^2 + 2f_1^3 + 2f_2^3 + f_1^4$

Figure 3 PMRES on the running example with modified, non-unit weights.

▶ **Example 7** (PMRES on a weighted formula)**.** Consider the running example, but with the modified weights $(1, 2, 3, 4, 5)$, respectively. We assume the same trail as shown in figure 2, and show in figure 3 the modified execution. After the first two iterations the lower bound will be 2, as shown. The optimum hitting set is $\{b_2, b_3\}$ with cost 5, so the lower bound does not match the optimum. Indeed, $H_R^2\mid_0$ is unsatisfiable: the clause $(f_2^1 \vee b_5)$ can only be satisfied by f_2^1, because $w^2(b_5) > 0$. But $f_2^1 \iff b_2 \wedge (b_3 \vee b_4)$ and $w^2(b_2) > 0, w^2(b_3) > 0, w^2(b_4) > 0$, therefore f_2^1 is forced to false. Hence, PMRES has to perform more iterations before matching the bound of the hitting set. A possible trail finds the metas $\{b_3, b_4, b_5\}$ and $\{b_2, b_5\}$ (which also happen to be cores of W^0), as shown.

We are now ready to state the main result of this section.

▶ **Theorem 8.** *For a PMS instance, at each iteration,* PMRES *computes an optimum hitting set of $\mathcal{C}_\cup^i$.*

Proof. Follows from Lemma 2, Corollary 5, and Lemma 6. ◀

For a WPMS instance, we can get a weaker result: since cores of $H_R^i\mid_0$ are also cores of $H^i\mid_0$, we can extract cores of $H_R^i\mid_0$, which are metas of W until it becomes satisfiable, at which point the bound is a hitting set of $\mathcal{C}_\cup^i$. It is not clear if that is a desirable thing to do from a performance perspective.

3.4 PMRES and Linear Programming

In this section, we prove the following.

▶ **Theorem 9.** *There exists an integer linear program ILP_P^i which (1) is logically equivalent to the minimum hitting set problem with sets $\mathcal{C}_\cup^i$, (2) has size polynomial in $|H_R^i|$, and (3) whose linear relaxation has an optimum which matches that derived by* PMRES*.*

Given the results of section 3.3, (1) is easy to show, since we can generate the set $\mathcal{C}_\cup^i$, then write the hitting constraint for each set in $\mathcal{C}_\cup^i$, and use w^0 as the objective. Call this ILP_{hs}^i. But ILP_{hs}^i may be exponentially larger than H_R^i. It is not much harder to show that we can achieve (1) and (2). As Corollary 5 shows, H_R^i is logically equivalent to that hitting set problem, so we can replace the constraints of ILP_{hs}^i by H_R^i (i.e., by the standard encoding of clauses to linear constraints) and get an equivalent problem. Call that ILP_R^i and its linear relaxation LP_R^i.

However, we can see that LP_R^i is weak, specifically, that $c(LP_R^i) < c(ILP_R^i)$.

▶ **Example 10** (Running example, continued)**.** Consider the ILPs ILP^2_{hs} and ILP^2_R corresponding to the hitting set problems for the 2^{nd} iteration of PMRES on the instance W in our running example. The optimum of both ILP^2_{hs} and ILP^2_R is 2, as expected, but the optimum of LP^2_R is only 1.5.

In this specific example, since we have integer costs, the bound of the linear relaxation allows us to derive a bound of 2 for ILP^2_R, but in general we can get an arbitrarily large difference. This is not surprising in general, but the fact that PMRES does compute an optimal hitting set at each iteration suggests that we should be able to do better. This is the objective of this section.

To construct an LP that meets the requirement of the theorem, we give a WCSP and its reformulation, which yield an LP (the local polytope) and a dual solution (one which is created from the formulation), as described in section 2.2. The result could be proved by directly giving an appropriate LP and dual solution, and proving the result on that, but it would be more cumbersome and would lack the existing intuitive understanding that has been developed in WCSP of dual solutions as reformulations.

Proof of theorem 9. We will give first a WCSP P^i which admits the same solutions as H^i_R and has unary costs such that its feasible solutions have the same cost as the hitting set problem entailed at iteration i of PMRES. This means that the optimum solution of P^i matches the minimum hitting set of $\mathcal{C}^i_\cup$. Further, we show that its linear relaxation $LP(P^i)$ admits a dual feasible solution whose cost matches the bound computed by PMRES. We give this dual solution as a sequence of equivalence preserving transformations of P^i, using the results presented in section 2.2. That linear program, $LP(P^i)$, satisfies the requirements of the theorem.

We first define P^i. The high level idea is that the we encode the objective function of ILP^i_R directly as unary costs, and each meta using the well known decomposition into ternary constraints. The d variables have exactly the same semantics as the auxiliary variables used in that decomposition. The corresponding f variable corresponds to a single tuple of these ternary constraints, so we add an f variable to each ternary constraint in order to capture the cost of that ternary tuple into a unary cost. More precisely, let $P^0 = \emptyset$. At iteration i, where the core discovered is $\{b^i_1, b^i_2, \ldots, b^i_r\} \subseteq S^{i-1}$, P^i is defined as P^{i-1} and additionally the following variables and cost functions:

- 0/1 variables $b_1, \ldots, b_n$, d^i_j, f^i_k, corresponding to the propositional variables of the same name in W^i.
- Unary cost functions with scope b_i for each $b_i \in vars(W^0)$, with $c_{b_i}(0) = 0, c_{b_i}(1) = c^0(b_i)$
- A ternary cost function with scope $\{b^i_1, d^i_1, f^i_1\}$ where each tuple that satisfies $b^i_1 \vee d^i_1$ and $f^i_1 \iff d^i_1 \wedge b^i_1$ has cost 0 and the rest have infinite cost.
- Quaternary cost functions with scope $\{b^i_j, d^i_{j-1}, d^i_j, f^i_j\}$, for $j \in [2, r-2]$, where each tuple that satisfies $d^i_{j-1} \iff d^i_j \vee b^i_j$ and $f^i_j \iff d^i_j \wedge b^i_j$ has cost 0 and the rest have infinite cost.
- A binary cost function with cost 0 for each tuple that satisfies $d^i_{r-1} = b^i_r$ and infinite cost otherwise.

It is straightforward to see that P^i is equivalent to ILP^i_R: (i) they have the same set of variables, (ii) the only costs in P^i are in unary cost functions, so the objective functions are the same, (iii) the quaternary cost functions satisfy, by construction, the clauses included in the scope of these functions, and (iv) each clause is present in one cost function. Therefore, solutions of P^i are hitting sets of $\mathcal{C}^i_\cup$ and the cost of each solution matches the cost of the corresponding hitting set.

b^1_1	d^1_1	f^1_1	⓪	①	②
0	1	0	0		
1	0	0	0	w	0
1	1	1	0	w	0

b^1_2	d^1_1	d^1_2	f^1_2	⓪	③	④
0	0	0	0	0	w	0
0	1	1	0	0		
1	1	0	0	0	w	0
1	1	1	1	0	w	0

b^1_1	⓪	①
0	0	
1	w	0

d^1_1	⓪	②	③
0	0	w	0
1	0		

f^1_1	⓪	②
0	0	
1	0	w

b^1_2	⓪	③
0	0	
1	w	0

b^1_3	⓪	④	⑤
0	0	w	0
1	w	w	0

f^1_2	⓪	④
0	0	
1	0	w

$c_\varnothing$	⓪	⑤
	0	w

Figure 4 The evolution of cost functions that leads to the increase of the lower bound by w for the core $\{b^1_1, b^1_2, b^1_3\}$. Each table shows a cost function and how it evolves after each EPT. We omit the rows which would violate one of the clauses introduced by PMRES, as infinity absorbs all costs, so they are unaffected by EPTs. The column ⓪ gives the initial costs. Subsequent columns give the state of each cost function after all EPTs to that point. Only points in the sequence which affect a given cost function are given in the corresponding table. Since $d^1_2 = b^1_3$ for this core, we simplify the problem here and replace occurrences of d^1_2 by b^1_3 rather than include an extra binary cost function to enforce their equality. The sequence is ①: $ext(b^1_1, \{b^1_1, d^1_1, f^1_1\}, w)$, ②: $prj(\{b^1_1, d^1_1, f^1_1\}, \overline{d}^1_1, w)$ and $prj(\{b^1_1, d^1_1, f^1_1\}, f^1_1, w)$, ③: $ext(b^1_2, \{b^1_{,} d^1_1, b^1_3, f^1_1\}, w)$ and $ext(\overline{d}^1_1, \{b^1_{,} d^1_1, b^1_3, f^1_1\}, w)$, ④: $prj(\{b^1_{,} d^1_1, b^1_3, f^1_1\}, \overline{b}^1_3, w)$ and $prj(\{b^1_{,} d^1_1, b^1_3, f^1_1\}, f^1_2, w)$, ⑤: $prj_0(b^1_3, w)$.

It remains only to show that the LP optimum of $relax(P^i)$ matches that produced by PMRES. We show a slightly stronger result, namely that there exists a sequence of EPTs such that in P^i, not only does the bound match that produced by PMRES, but the unary costs of each variable match the weights computed by PMRES. We show this by induction on the number of iterations. At iteration 0, this holds trivially, as the bound is 0 for both P^0 and PMRES and the unary costs match the weights by construction. Suppose it holds at iteration $k-1$. Then, the core at iteration k is $\{b^k_1, b^k_2, \ldots, b^k_r\} \subseteq S^{k-1}$. The EPT $ext(b^k_1, \{b^k_1, d^k_1, f^k_1\}, w^k_{\min})$ enables the EPTs $prj(\{b^k_1, \overline{d}^k_1, f^k_1\}, f^k_1, w^k_{\min})$ and $prj(\{b^k_1, d^k_1, f^k_1\}, \overline{d}^k_1, w^k_{\min})$. For $j \in [2, r^k - 2]$, in addition to extending cost from b^k_j, we also extend from $\overline{d}^k_{j-1}$, which has just received this amount of cost: $ext(b^k_j, \{b^k_j, d^k_{j-1}, d^k_j, f^k_j\}, w^k_{\min})$ and $ext(\overline{d}^k_{j-1}, \{b^k_j, \overline{d}^k_{j-1}, d^k_j, f^k_j\}, w^k_{\min})$, which enable $prj(\{b^k_j, d^k_{j-1}, d^k_j, f^k_j\}, f^k_j, w^k_{\min})$ and $prj(\{b^k_j, d^k_{j-1}, d^k_j, f^k_j\}, \overline{d}^k_j, w^k_{\min})$. Finally, after $j = r - 2$, we are left with $w^k_{\min}$ in $\overline{d}^k_{r-1}$. Using $d^k_{r-1} \iff b^k_r$, we move cost from b^k_r to d^k_{r-1} (specifically: $ext((, b^k_r, \{b^k_r, d^k_r\})w^k_{\min}$, then $prj(\{b^k_r, d^k_r\}, d^k_r, w^k_{\min})$. Since both d^k_r and $\overline{d}^k_r$ have cost $w^k_{\min}$, we can apply $prj_0(d^k_r, w^k_{\min})$ to increase the lower bound by $w^k_{\min}$.

After these EPTs, not only is the lower bound increased by $w^k_{\min}$, but the variables $b^k_1, \ldots, b^k_r$ have their cost decreased by $w^k_{\min}$, the variables $f^k_1, \ldots, f^k_{r-1}$ receive cost $w^k_{\min}$, and the variables $d^k_1, \ldots, d^k_{r-1}$ stay at 0. This matches the effects of PMRES, as required by the inductive hypothesis. ◀

▶ **Example 11.** We move away from our running example here, as showing and explaining all the cost moves would be tedious and space consuming. Instead, we give a small example with the core $\{b^1_1, b^1_2, b^1_3\}$ in figure 4. All variables of this core have uniform weight w. We show how the EPTs remove cost from b^1_1, b^1_2, b^1_3 and move it to f^1_1, f^1_2 and $c_\varnothing$, leaving all other cost functions unchanged, even though they were used to make the cost moves possible. The increase in $c_\varnothing$ comes from a nullary projection from b^1_3.

Note that theorem 9 does not prove that the optimum of (P^i) is identical to that of PMRES at iteration i, but only that it is at least as high, as the following example shows.

▶ **Example 12** (Running example, continued)**.** After iteration 2, in the running example, unit propagation alone detects the core $\{b_3, b_4, b_5\}$. This means that when we set these variables to false because their weight is non-zero, unit propagation generates the empty clause.

Let $\hat{P}^i$ be the reformulation of P^i given by theorem 9. Then H^i_R and $\hat{P}^i$ have the same costs/weights. $H^i_R \mid_0$ is constructed from H^i_R in the same way as $Bool(P^i)$ is constructed from $Bool(P)$: by making each non zero cost (weight) into an infinite cost (weight). so $H^i_R \mid_0$ admits the same solutions as $Bool(\hat{P}^i)$. Moreover, each clause of $H^i_R \mid_0$ is contained in at least one constraint of $\hat{P}^i$, therfore arc consistency on $Bool(\hat{P}^i)$ is at least as strong as unit propagation on $H^i_R \mid_0$. And since the core $\{b_3, b_4, b_5\}$ is not satisfied, the arc consistency closure of $Bool(\hat{P}^i)$ is empty, therefore its bound can be improved further.

On the other hand, there is no reason to expect that the the optimum of (P^i) will necessarily be higher than the bound computed by PMRES. For example, if $H^i_R \mid_0$ has no cores that can be detected by unit propagation, the argument of example 12 does not apply.

4 OLL

OLL [22] is probably the most relevant core-guided algorithm currently, since solvers based on it, like RC2 [19] and CASHWMaxSAT-CorePlus [21] have done very well in recent MaxSAT evaluations [5].

4.1 MaxSAT with soft cardinality constraints

OLL is an iterative algorithm, similar to PMRES. For the purposes of this discussion, it only differs in how it processes each meta that it finds. At iteration i, given the meta $m^i = \{b^i_1, b^i_2, \ldots, b^i_{r^i}\} \subseteq S^{i-1}$, it adds fresh variables $o^i_1, \ldots, o^i_{r^i-1}$ and constraints $o_j \iff \sum_{k=1}^{r^i} b^i_k > j$, then decreases the weight of each variable in m^i by w^i_{min}, increases the lower bound by w^i_{min}, and sets the weight of the fresh variables $o^i_1, \ldots, o^i_{r^i-1}$ to w^i_{min}. The o variables are called *sum variables.*

OLL with implied cores

We use here a minor modification of OLL, which we denote OLL′. In this variant, before processing a meta at iteration i, each sum variable o^j_k, $j < i, k \in [2, r^j - 1]$ is replaced by $o^j_{k'}$ where $k' < k$ is the lowest index for which $w(o^j_{k'}) > 0$. This is sound because $o^j_k \to o^j_{k'}$ for all $k' < k$, which can be written as $\neg o^j_k \vee o^{k'}$. We can resolve the meta at iteration i with this clause to effectively replace o^j_k by $o^j_{k'}$. This procedure can be repeated as long as it results in a meta with non-zero minimum weight, although that step is not required for the results we obtain next.

We argue that OLL′ matches the behaviour of a realistic implementation like RC2, when used with an assumption-based solver such as MINISAT [16] or a derivative like GLUCOSE [4]. In order to extract a core with MINISAT, RC2 asserts the negation of all literals which may appear in a core as assumptions. These literals are passed to MINISAT as a sequence. MINISAT returns a subset of these literals as a core. Crucially, MINISAT immediately propagates each assumption in sequence and never returns in a core a literal which is implied by earlier assumptions. Therefore, if the literals of the soft clauses introduced by OLL are given in the order $\langle o^i_1, \ldots o^i_{r^i} \rangle$, we get from $o^i_{j+1} \implies o^i_j$, or equivalently $\overline{o}^i_j \implies \overline{o}^i_{j+1}$, that all literals $\overline{o}^i_j$

are implied by unit propagation from $o^i_{j'}$ with $j' < j$. Therefore, MINISAT will not return a core that contains o^i_j if $o^i_{j'}$ is in the assumptions. This means that OLL$'$ is identical to OLL given these implementation details. By inspection of the code of RC2, we can confirm that it does indeed use this order of assumptions with MINISAT, and therefore implements OLL$'$.

4.2 Cores and Hitting Sets of OLL

In the following, we overload notation that we have used already for PMRES, but we use them now in the context of OLL$'$, with the same meaning: $H^i_R, \mathcal{C}^i, \mathcal{C}^i_\cup$.

▶ **Lemma 13.** *There exists a set $\mathcal{C}^i$ such that m^i is a core of H^i if and only if for each $c \in \mathcal{C}^i$, c is a core of ϕ.*

Proof Sketch. We observe that $o^i_j = \bigvee_{S \subseteq m^i, |S| > j}(\wedge_{b \in S} b)$, therefore it is a monotone function of the inputs of the core. The entire formula constructed by OLL is therefore also monotone. We show the result using a similar variable forgetting argument as we did in lemma 2. ◀

The proofs of Lemma 3, Observation 4, and Corollary 5 transfer to OLL$'$ immediately. These establish that the WPMS instance $\langle H^i_R, cost^i \rangle$ encodes the minimum hitting set problem over $\mathcal{C}^i_\cup$, where the cores are derived as described in lemma 13 this time.

In order to show that OLL$'$ does compute minimum hitting sets at each iteration for PMS, we have to prove the equivalent of lemma 6.

▶ **Lemma 14.** *If W is a PMS instance, $H^i_R \mid_0$ is satisfiable for all iterations i of* OLL$'$.

Proof Sketch. The following invariant holds in OLL$'$: for each meta m^i, there exists $0 \leq k < r^i$ such that $w(o^i_{k'}) = 0$ for all $k' \leq k$ and $w(o^i_{k'}) > 0$ for all $k' > k$. Therefore, any assignment that sets $o^j_{k'}$, $k' < k$, to true can be extended by setting $o^j_{k''}$ to true as well for all $k'' < k'$ and exactly k' variables of m^i, so that all sum constraints of iteration i are satisfied.

From there, we use the same argument as we did in the proof of lemma 6 to show that, given an assignment to the variables of the metas $m^j, \ldots, m^i$, $j < i$, we can extend to an assignment to the variables of m^{j-1} because any two sum constraints from different iterations sum over disjoint sets of variables. ◀

As was the case for the corresponding lemma in PMRES, Lemma 14 says nothing about instances with non-uniform weights.

4.3 OLL and Linear Programming

We prove the equivalent of theorem 9 for OLL$'$.

▶ **Theorem 15.** *There exists an integer linear program ILP^i_P which (1) is logically equivalent to the minimum hitting set problem with sets $\mathcal{C}^i_\cup$, (2) has size polynomial in $|H^i_R|$, and (3) whose linear relaxation has an optimum which matches that derived by* OLL$'$.

Proof. We construct a WCSP P^i. Its linear relaxation, the local polytope $LP(P^i)$, is the LP we want. Let $P^0 = \emptyset$. At iteration i, where the core discovered is $\{b^i_1, b^i_2, \ldots, b^i_{r^i}\} \subseteq S^{i-1}$, P^i is defined as P^{i-1} and additionally the following variables and cost functions:

- 0/1 variables $b^i_1, \ldots, b^i_n$, $o^i_1, \ldots, o^i_{r^i-1}$, corresponding to the propositional variables of the same name in W^i.
- Unary cost functions with scope b_i for each $b_i \in vars(W^0)$, with $c(b_i, 0) = 0, c(b_i, 1) = c^0(b_i)$

- A variable O^i with domain $[0, r^j]$, with $c(O^i, 0) = \infty$ and $c(O^i, j) = 0$ for all $j \in [1, r^i]$.
- A decomposition of the sum constraint $\sum_{j\in[1,r^i]} b^i_j = O^i$, as described by Allouche et al. [1].
- Binary cost functions with scope $\{O^i, o^i_j\}$, for all $j \in [1, r^i - 1]$ where the tuples $\{j', 1\}$ and $\{j'', 0\}$, for all $1 \leq j' < j < j'' < r^i$, have infinite cost, and the rest have cost 0. These encode the constraint $o^i_j \iff O^i > j$.

As before, the equivalence of P^i and H^i_R is immediate. We show that there exists a reformulation of P^i that yields the same costs as the weights computed by OLL′, as well as the same lower bound. The latter relies on previous results [1], which imply that, we can move cost w^i_{min} from $b^i_1, \ldots, b^i_n$ to O^i, so that we have $c(O_i, j) = jw^i_{\text{min}}$. Since $c(O^i, 0) = \infty$, we can apply $prj_0(O^i, w^i_{\text{min}})$. Finally, we can apply $ext(O^i = j', \{O^i, o^i_j\}, w^i_{\text{min}})$ for all $j' \geq j$, followed by $prj(\{O^i, o^i_j, o^i_j, w^i_{\text{min}})$. Once we complete this for all $j \in [1, r^i]$, there is no cost in O^i, and each o^i_j has cost w^i_{min}, as required. ◀

5 Connection to the Sherali-Adams hierarchy

The Sherali-Adams hierarchy of linear relaxations [28] of a 0/1 integer linear program is a well known construction for building stronger relaxations. At its k^{th} level, it uses monomials of degree k and it is known that the level n relaxation (where n is the number of variables in the ILP) represents the convex hull of the original ILP, meaning that it solves the ILP exactly. On the flip side, the size of the relaxations grows exponentially with the level of the hierarchy, meaning that even low level SA relaxations tend to be impractical.

Formally, we derive the k^{th} level SA relaxation as follows. Let $SA^u_0(LP) = LP$, the linear relaxation of the integer program. First, we define the set of multipliers $M_k = \{\prod_{i\in P_1} x_i \prod_{i\in P_2}(1 - x_i) \mid P_1, P_2 \subseteq [1, n], |P_1 \cup P_2| = k, P_1 \cap P_2 = \emptyset\}$, i.e., the set of all non-tautological monomials of degree k, using either x_i or $(1 - x_i)$ as factors. We then multiply each constraint $c \in LP_0$ by each multiplier $m \in M_k$, simplify using $x^2 = 1$ and $x(1 - x) = 0$, and finally we replace each higher order monomial by a single 0/1 variable to get $SA^u_k(LP)$.

In this description, SA^u_k does not contain the variables and constraints of LP or any SA^u_j, $j \in [1, k - 1]$. Here, we use instead $SA_k(LP) = \cup^k_{i=0}(SA^u_k(LP) \cup cns(k)$, where $cns(k)$ are constraints which ensure consistency between the variables at different levels, i.e., do not allow $x_ix_j = 1$ and $x_i = 0$ at the same time.

To show the connection with PMRES, we define the *depth* measure for variables and, by extension, cores and formulas. The set the depth of all variables appearing in W^0 to be 0, and we write $dp(b_j) = 0$, for $b_j \in vars(W^0)$. Consider a meta m^i. We define $dp(f^i_j) = \max_{b_j\in m^i} dp(b_j) + 1$ for all $j \in [1, r^i - 1]$, and similarly for $d^i_j, j \in [1, r^i - 1]$. With an overload of notation, we also write $dp(m^i) = dp(f^i_1)$. Finally, at iteration i, we write $dp(W^i) = \max_{j\in[1,i]} dp(m^j)$. In words, the depth of a variable of the original instance has depth 0, the variables introduced by a meta are one level deeper than variables that appear in the meta, the depth of a meta is the same as that of the variables it introduces, and the depth of the instance at iteration i is the deepest meta PMRES has discovered.

The result of this section, is that $LP(P^i)$, the linear relaxation that achieves the bound computed by PMRES, is a subset of the $2^{dp(W^i)}$ level Sherali-Adams relaxation of a specific linear formulation of the hitting set instance $\mathcal{C}^i_\cup$.

▶ **Theorem 16.** *The variables f^i_j with $dp(f^i_j) = k$ are defined as a linear expression over variables of at most the level 2^k SA relaxation of the hitting set problem over $\mathcal{C}^i_\cup$.*

Proof. By induction. It holds for variables with depth 0, since they are variables of the original formula. Assume that it holds for variables of depth $k-1$.

The main observation is that, since $f^i_j = b^i_j \wedge d^i_j$, we can write it as $f^i_j = b^i_j d^i_j$, i.e., replace the conjunction by multiplication, which is valid for 0/1 variables. Then, since $d^i_j = b^i_{j+1} \vee \ldots \vee b^i_{r^i}$, we can write it as $d^i_j = \max(b^i_{j+1}, \ldots, b^i_{r^i})$. The max operator is a piecewise linear function, so this expression is linear. Finally, we replace d^i_j in the definition of f^i_j to get $f^i_j = \max(b^i_j b^i_{j+1}, \ldots, b^i_j b^i_{r^i})$. Recall that $dp(b^i_l)$ for $l \in [j+1, r^i]$ is at most 2^{k-1}, so f^i_j can be written as a linear expression over monomials of degree at most 2^k, since it multiplies two variables which are themselves a linear expression over monomials of degree at most 2^{k-1}. ◀

Theorem 16 reflects the already known connection between Max-Resolution and the Sherali-Adams hierarchy in the context of proof systems for satisfiability [17]. Moreover, it is known that the k^{th} level of the Sherali Adams hierarchy based on the *basic LP relaxation* (BLP) of a CSP, another name for the local polytope LP, establishes k-consistency [29].

Theorem 16 is fairly weak. The upper bound is extremely loose and there is no lower bound. It is useful, however, as it suggests that discovering a meta of depth k involves potentially proving 2^k-inconsistency. It also hints towards minimizing the maximum degree of monomials entailed by a meta as a metric for choosing among different potential metas.

In the greater context of PMRES compared to IHS, one way to interpret the result of this section is that the two algorithms are instantiations of the same algorithm: they are both implicit hitting set algorithms, but where IHS extracts a single core at a time and offloads the hitting set computation to a specialized solver, PMRES shifts the burden to the SAT solver to not only extract cores, but discover a higher level relaxation so that the hitting set problem can be solved in polynomial time.

6 Discussion

6.1 PM1

The results of section 3.3 have of course already been shown for PM1 [7, 25]. The result we have shown here that is not shown for PM1 is the existence of a compact LP that computes the same bound as PM1. It is not easy to see how the results of section 3.4 could transfer. For PMRES and OLL, H^i_R logically entails all the implied cores. This allows us to create an ILP representation of the hitting set problem immediately, and then strenghten the LP relaxation using higher order cost functions to achieve the same bound. But for PM1, cores are solutions of a linear system, so it is not immediately obvious even how to create an ILP representation of the hitting set problem without enumerating the (potentially exponentially many) cores of the original formula.

6.2 Practical Implications

Besides revealing a tight connection between the operation of IHS and core-guided algorithms, there are potential practical implications, in particular from theorem 9. We first observe that the linear program used to prove theorem 9 is linear in the size of H^i_R, hence the size of the LP is not too great. Moreover, it can be further reduced by noting that, in order to replicate the bound of PMRES, the dual variable corresponding to several primal constraints is always zero. Therefore, they can be removed from the LP without affecting the bound. After that, the LP can be further simplified by removing variables that appear in only 1 constraint and forgetting (in the sense of the knowledge compilation operation of forgetting) variables that appear in only two constraints. In this way, the LP is reduced to contain only the d and f variables, and uses r^i constraints to relate them. In the running example, upon discovering the core $\{b_1, b_2, b_3, b_4\}$, the LP needs only the following constraints to satisfy the requirements of theorem 9:

$$\begin{aligned} b^1_1 - f^1_1 - d^1_1 &= 0 \\ b^1_2 - f^1_2 - d^1_2 + d^1_1 &= 0 \\ b^1_3 - f^1_3 + b^1_4 + d^1_2 &= 1 \end{aligned}$$

We omit the details of this mechanical reduction of the LP. But this suggests that the LP of theorem 9 is not just a theoretical construct, but a practical way to replicate the reasoning of PMRES. This allows a solver which runs PMRES until some heuristic condition is met, then passes its progress to IHS using theorem 9 to represent the hitting set problem and the lower bound. In the other direction, a solver can run IHS, then solve the hitting set problem once with PMRES to construct H^i_R, then continue solving starting from $\langle H^i_R \cup H, w^i \rangle$, in order to simplify solution of the ILP. However, running the two algorithms in sequence is the simplest form of combining them. Presumably, the greatest performance can be gained by an even deeper integration, using the LP to communicate progress.

7 Conclusion

We have narrowed the gap between implicit hitting set and core-guided algorithms for MaxSAT. We have shown that the core-guided algorithms PMRES and OLL, the latter of which is the basis for the winning solvers of some recent maxsat evaluations, implicitly compute a potentially exponentially large set of cores of the original MaxSAT formula at each iteration and a minimum hitting set of those cores under some conditions. Moreover, we showed that they build a WPMS instance which is logically equivalent to the minimum hitting set problem over those cores and can therefore be seen as a compressed, polynomial sized, encoding of that problem. In addition, we showed how this problem is solved: by generating a subset of a higher level of the Sherali-Adams linear relaxation of that hitting set problem. These results open up the possibility for tighter integration between PMRES and IHS.

References

1 David Allouche, Christian Bessiere, Patrice Boizumault, Simon de Givry, Patricia Gutierrez, Jimmy H. M. Lee, Ka Lun Leung, Samir Loudni, Jean-Philippe Métivier, Thomas Schiex, and Yi Wu. Tractability-preserving transformations of global cost functions. *Artif. Intell.*, 238:166–189, 2016. doi:10.1016/j.artint.2016.06.005.

2 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *International conference on theory and applications of satisfiability testing*, pages 427–440. Springer, 2009.

3 Carlos Ansótegui and Joel Gabàs. WPM3: an (in)complete algorithm for weighted partial maxsat. *Artif. Intell.*, 250:37–57, 2017. doi:10.1016/j.artint.2017.05.003.

4 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the International Joint Conference on Artifical Intelligence (IJCAI)*, pages 399–404, 2009.

5 F. Bacchus, J. Berg, M. Järvisalo, R. Martins, and A. (eds) Niskanen. MaxSAT evaluation 2022: Solver and benchmark descriptions. Technical Report vol. B-2022-2, Department of Computer Science, University of Helsinki, Helsinki, 2022. URL: http://hdl.handle.net/10138/318451.

6 Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in maxsat. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming – 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017. doi:10.1007/978-3-319-66158-2_41.

7 Fahiem Bacchus and Nina Narodytska. Cores in core based maxsat algorithms: An analysis. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014 – 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 7–15. Springer, 2014. doi:10.1007/978-3-319-09284-3_2.

8 Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set maxsat solving. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020. doi:10.1007/978-3-030-51825-7_20.

9 Jeremias Berg, Paul Saikko, and Matti Järvisalo. Improving the effectiveness of sat-based preprocessing for maxsat. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 239–245. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/040.

10 Maria Luisa Bonet, Jordi Levy, and Felip Manyà. A complete calculus for max-sat. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing – SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 240–251. Springer, 2006. doi:10.1007/11814948_24.

11 M. C. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8):449–478, May 2010. doi:10.1016/j.artint.2010.02.001.

12 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming – CP 2011 – 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011. doi:10.1007/978-3-642-23786-7_19.

13 Jessica Davies and Fahiem Bacchus. Exploiting the power of mip solvers in maxsat. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing – SAT 2013 – 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013. doi:10.1007/978-3-642-39071-5_13.

14 Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming – 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013. doi:10.1007/978-3-642-40627-0_21.

15 Tomás Dlask and Tomás Werner. On relation between constraint propagation and block-coordinate descent in linear programs. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming – 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 194–210. Springer, 2020. doi:10.1007/978-3-030-58475-7_12.

16 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of Theory and Applications of Satisfiability Testing (SAT)*, pages 502–518, 2003.

17 Yuval Filmus, Meena Mahajan, Gaurav Sood, and Marc Vinyals. Maxsat resolution and subcube sums. *ACM Trans. Comput. Logic*, 24(1), January 2023. doi:10.1145/3565363.

18 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing – SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006. doi:10.1007/11814948_25.

19 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi:10.3233/SAT190116.

20 Javier Larrosa and Federico Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 – August 5, 2005*, pages 193–198, 2005.

21 Zhendong Lei, Yiyuan Wang, Shiwei Pan, Shaowei Cai, and Minghao Yin. CASHWMaxSAT-CorePlus: Solver description. Technical report, Department of Computer Science, University of Helsinki, Helsinki, 2022. URL: http://hdl.handle.net/10138/318451.

22 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided maxsat with soft cardinality constraints. 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 564–573. Springer, 2014. doi:10.1007/978-3-319-10428-7_41.

23 António Morgado, Alexey Ignatiev, and João Marques-Silva. MSCG: robust core-guided maxsat solving. *J. Satisf. Boolean Model. Comput.*, 9(1):129–134, 2014. doi:10.3233/sat190105.

24 Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2717–2723, 2014. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8513.

25 Nina Narodytska and Nikolaj S. Bjørner. Analysis of core-guided maxsat using cores and correction sets. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 26:1–26:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAT.2022.26.

26 Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2.

27 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid maxsat solver. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016. doi:10.1007/978-3-319-40970-2_34.

28 Hanif D. Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990. doi:10.1137/0403036.

29 Johan Thapper and Stanislav Zivný. Sherali-adams relaxations for valued csps. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 1058–1069. Springer, 2015. doi:10.1007/978-3-662-47672-7_86.

A SAT Solver's Opinion on the Erdős-Faber-Lovász Conjecture

Markus Kirchweger
Algorithms and Complexity Group, TU Wien, Austria

Tomáš Peitl
Algorithms and Complexity Group, TU Wien, Austria

Stefan Szeider
Algorithms and Complexity Group, TU Wien, Austria

Abstract

In 1972, Paul Erdős, Vance Faber, and Lászlo Lovász asked whether every linear hypergraph with n vertices can be edge-colored with n colors, a statement that has come to be known as the *EFL conjecture*. Erdős himself considered the conjecture as one of his three favorite open problems, and offered increasing money prizes for its solution on several occasions. A proof of the conjecture was recently announced, for all but a finite number of hypergraphs. In this paper we look at some of the cases not covered by this proof.

We use SAT solvers, and in particular the SAT Modulo Symmetries (SMS) framework, to generate non-colorable linear hypergraphs with a fixed number of vertices and hyperedges modulo isomorphisms. Since hypergraph colorability is NP-hard, we cannot directly express in a propositional formula that we want only non-colorable hypergraphs. Instead, we use one SAT (SMS) solver to generate candidate hypergraphs modulo isomorphisms, and another to reject them by finding a coloring. Each successive candidate is required to defeat all previous colorings, whereby we avoid having to generate and test all linear hypergraphs.

Computational methods have previously been used to verify the EFL conjecture for small hypergraphs. We verify and extend these results to larger values and discuss challenges and directions. Ours is the first computational approach to the EFL conjecture that allows producing independently verifiable, DRAT proofs.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Mathematics of computing → Hypergraphs; Mathematics of computing → Graph coloring; Mathematics of computing → Graph enumeration; Mathematics of computing → Matchings and factors

Keywords and phrases hypergraphs, graph coloring, SAT modulo symmetries

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.13

Supplementary Material *Software (Source Code)*: `https://github.com/markirch/sat-modulo-symmetries/`, archived at `swh:1:dir:c50f1ce7d22717e113b915ae9976da0cb0e67c1d`
Software (Documentation): `https://sat-modulo-symmetries.readthedocs.io/`

Funding This work was supported by the Austrian Science Fund (FWF, project P32441), and the Vienna Science and Technology Fund (WWTF, project ICT19-065).

1 Introduction

In 1972, Paul Erdős and László Lovász gathered at Vance Faber's apartment for some tea, and to prove what was supposed to be an easy theorem about hypergraph edge colorings: every linear hypergraph can be edge-colored with no more colors than it has vertices. A hypergraph is a collection of subsets, called (hyper)edges, of a finite set, and it is linear if any two hyperedges intersect in at most one element (vertex). An edge coloring is a coloring of the edges such that intersecting edges have different colors. The conjecture, now known as

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 13; pp. 13:1–13:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

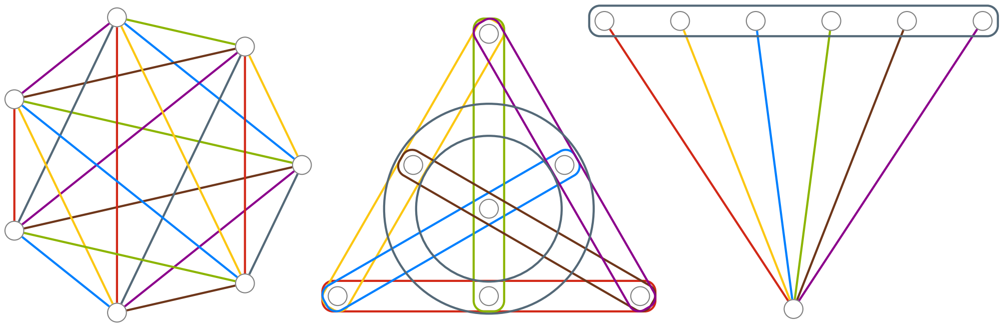


Figure 1 The famous extremal examples to the EFL conjecture for $n = 7$. Left to right: the complete graph, the projective plane of order 2 (also known as the Fano plane), and the degenerate plane. Hyperedges are drawn as bounding boxes around vertices, 2-edges are drawn simply as lines. Each hypergraph is colored with its optimal number of colors; in the case of the planes this is trivial as the number of colors equals the number of hyperedges.

the Erdős-Faber-Lovász conjecture (EFL) conjecture, is inspired by its prominent extremal examples, the complete graph, the degenerate plane, and the projective plane (see Figure 1), each of which requires the full number of colors.

Needless to say, they did not find an easy proof, and the conjecture has remained open for fifty years. Erdős wrote about it as one of his favorite combinatorial problems [7] and offered money prizes for a solution [6]. Throughout its lifetime, the conjecture attracted a lot of attention, and many partial results have been proved. The newest in the series is the recently announced proof of the conjecture for all but a finite number of cases [17].

In this paper, we look at the conjecture by computational means. We investigate classes of small hypergraphs, and verify that they can indeed be colored with the conjectured number of colors. In this respect we follow in the footsteps of Hindman [12] and Romero and Alonso-Pecina [22], who also investigated small classes of hypergraphs and partially verified the EFL conjecture. The novel aspects of our work are:

1. we verify the EFL conjecture for more classes of small hypergraphs;
2. where the previous two works had to enumerate and afterwards color all linear hypergraphs, we interleave these two processes: we generate hypergraphs and colorings alternately, and thus our method is not constrained by the number of linear hypergraphs, but only by the number of colorings needed to color them all. In doing this we build on the recently-proposed *co-certificate learning (CCL)* [18].
3. We use SAT encodings and SAT solvers, and hence our method enables us to produce independently verifiable proofs, including of the previous computational results. We are able to use them effectively due to the recently introduced *SAT modulo symmetries (SMS)* framework [20], in which a CDCL SAT solver [9] is enhanced with a custom symmetry propagator which prunes non-canonical search paths. In this work, we extend SMS, which was previously used for graphs, and later matroids, to hypergraphs. We achieve this by modeling hypergraphs by their bipartite incidence graphs, which in turn can be fit into the SMS framework with some care.

We report on the results of three experiments.

In the first, we tackle the EFL conjecture itself and analyze the nature of the hardness of verifying the conjecture experimentally in cases we processed. Naturally, we have not found any counterexamples (the title of the paper would otherwise have been different).

In the second experiment, we search for extremal examples to the EFL conjecture, namely linear hypergraphs with n vertices that require n colors. In addition to the known extremal examples, we were able to enumerate all other small extremal hypergraphs. Inspired by our findings, we also proved extremality of some infinite families of linear hypergraphs by hand.

In the last experiment, we tackle a generalization of the EFL conjecture (Conjecture 3), which at the same time also generalizes Vizing's Theorem about edge colorings in graphs. This experiment proceeds in a very similar way as the first, and we did not find any counterexamples in this case either.

In the next sections, we first review the necessary background on hypergraphs and colorings, followed by an explanation of our SAT-based process, and a discussion of results.

2 The EFL Conjecture

For positive integers n, m, let $[n] = \{1, \dots, n\}$ and $[n, m] = \{n, n+1, \dots, m\}$. A hypergraph $H = (V, E)$ consists of a finite vertex set V (we assume $V = [n]$), and a set E of subsets of V called *hyperedges*, or sometimes simply *edges*. A hyperedge of size k is also called a *k-edge*. A hypergraph where each hyperedge has size 2 is called a *graph*. The graph, denoted by K_n, which contains all possible 2-edges between n vertices is called the *complete graph*, or the *clique*. The *degree* of a vertex is the number of hyperedges that contain v. A hypergraph is called *linear* if any two of its hyperedges intersect in at most one vertex. Every graph is trivially a linear hypergraph.

A *χ-edge coloring* of a hypergraph $H = (V, E)$ is a mapping $c : E \to [\chi]$ which colors intersecting hyperedges with different colors: $e \cap f \neq \emptyset \implies c(e) \neq c(f)$. The *chromatic index* of a hypergraph is the smallest integer χ for which a χ-edge coloring exists. The *Erdős-Faber-Lovász (EFL) conjecture* postulates the existence of certain edge colorings.

▶ **Conjecture 1** (EFL conjecture). *Every linear hypergraph with n vertices is n-edge-colorable.*

While this paper is primarily about edge colorings of hypergraphs, we will also need to work with vertex colorings of graphs. A *χ-coloring* of a graph $G = (V, E)$ is a mapping $c : V \to [\chi]$ that maps adjacent vertices to different colors: $\{u, v\} \in E \implies c(u) \neq c(v)$. The *chromatic number* of a graph is the smallest χ for which a χ-coloring exists.

A hypergraph H is *covered* if every pair of vertices $\{u, v\} \subseteq V$ is contained in some hyperedge $e \in E$. Linear hypergraphs which are covered and whose hyperedges have size at least two are known as *linear spaces*.

▶ **Observation 2.** *If there is a counterexample to the EFL Conjecture with n vertices, then there is a counterexample with n vertices which is a linear space.*

Indeed, one can simply delete singleton hyperedges without decreasing the chromatic index, due to the fact that singleton hyperedges intersect at most $n - 1$ other hyperedges in a linear hypergraph, and any n-edge coloring can always be extended to them. Similarly, adding a 2-edge to cover a pair $\{u, v\}$ cannot decrease the chromatic index either. By Observation 2, we thus restrict our search for counterexamples to the EFL conjecture to linear spaces.

The EFL conjecture, if true, is tight, as witnessed by the examples of the (odd) complete graphs, the degenerate planes, and the projective planes, each of which requires n colors.

When restricted to graphs, the EFL conjecture is easy to prove. The only linear space on n vertices is the complete graph, and it is an instructive exercise to verify that it is indeed n-colorable. In fact, a stronger result known as Vizing's Theorem holds for graphs: a graph of maximum degree Δ can be $(\Delta + 1)$-edge-colored (for these graphs we can no longer assume they are covered). Together with the trivial bound arising from the fact that the edges around a vertex of degree Δ require Δ different colors, Vizing's Theorem implies that the chromatic index of any graph is either Δ or $\Delta + 1$. Again, it is easy to verify that Vizing's Theorem in terms of the maximum degree does not hold for linear hypergraphs in general, but a different, beautiful generalization has been conjectured. This generalization arises from a reformulation of Vizing's Theorem in terms of the *closed neighborhood* of a vertex, which is the union of the hyperedges containing a vertex, $\bigcup_{v\in e} e$. In the case of graphs, the size of the closed neighborhood of v is, by definition, the degree of v + 1, and the maximum size of any closed neighborhood is $\Delta + 1$. Vizing's Theorem then says that a graph can be edge-colored with a number of colors equal to the largest size of a closed neighborhood. Now, *this* version has been conjectured for linear hypergraphs in general.

▶ **Conjecture 3** (Füredi [10], Berge [1]). *Let $H = (V, E)$ be a linear hypergraph, and let $\zeta = \max_{v\in V} \left|\bigcup_{v\in e} e\right|$. Then H is ζ-edge-colorable.*

We call this conjecture the FB Conjecture.

In the context of this conjecture, we can no longer assume that hypergraphs are covered, because arbitrary edge addition could increase neighborhood size beyond the admissible bound. Instead, we say a hypergraph is *weakly covered* if each vertex pair $\{u, v\} \subseteq V$ either is contained in some hyperedge $e \in E$ or one of the two vertices has maximum closed neighborhood. We state a similar observation as before:

▶ **Observation 4.** *If there is a counterexample to the FB Conjecture with n vertices, then there is a weakly covered counterexample with n vertices without hyperedges of size 1.*

Note how the two formulations of Vizing's Theorem coincide for graphs, but in presence of larger hyperedges the difference between counting and unifying the neighborhood materializes.

In our experiments, we will validate both conjectures, and we will also search for extremal examples in addition to those listed in Figure 1.

2.1 Previous Work

A large body of work on the EFL conjecture (Conjecture 1) is available, roughly categorizable into two groups. In the first group, there are the asymptotic results [3, 13–16, 23], of which the culmination is the recently announced proof of Conjecture 1 for all hypergraphs with a sufficiently large number of vertices [17]. If correct, this result, at least in theory, leaves the complete verification of the conjecture down to a finite computation, the verification of the colorability of a finite number of linear spaces.

In the second group, there are the results which verify the conjecture for a finite number of small hypergraphs. The first such result known to us is due to Hindman [12], who verified Conjecture 1 for all hypergraphs with at most 10 vertices. In fact, Hindman proved a stronger result, namely that every hypergraph whose hyperedges of size at least 3 span at most 10 vertices is colorable; and thus in particular those with at most 10 vertices in total.

▶ **Theorem 5** (Hindman [12]). *Let $H = (V, E)$ be a linear hypergraph with n vertices, let $E' = \{e \in E : |e| \geq 3\}$. If $|\bigcup E'| \leq 10$, then H is n-edge colorable.*

Later, Romero and Alonso-Pecina, building on an existing program to enumerate linear spaces [2], verified Conjecture 1 up to $n \leq 12$ [22]; their contribution lied mainly in a meta-heuristic coloring algorithm. In our work, we replace ad-hoc hypergraph generation and coloring tools with a SAT-based workflow, which is more flexible, reliable, and easier to use. We verify the existing results and explore some generalizations and larger cases.

3 Reduction to SAT

In the sequel, we shall implicitly generate linear hypergraphs with n vertices and check that they are colorable with the right number of colors χ in order to validate Conjectures 1 and 3. For the former conjecture, we set $\chi = n$, for the latter, χ is the size of the largest closed neighborhood. Since in each run we will generate hypergraphs with a fixed number of hyperedges, we first need to determine bounds on the number m of hyperedges to be considered for a given number of vertices n.

We can assume the number of hyperedges is at least $\chi + 1$, as $\leq \chi$ hyperedges are trivially colorable by χ colors, though for generating extremal examples, we also need to consider $m = \chi$, so we take $m \geq \chi$. On the upper side, since each hyperedge covers at least one pair of vertices and no pair is covered twice (which is the same as linearity), the largest possible number of hyperedges is $\binom{n}{2}$, when the hypergraph is the complete graph on n vertices.

Additionally, we impose the restrictions given by Observation 2 and 4 respectively.

To enumerate linear hypergraphs with n vertices and m hyperedges, we produce a propositional formula $\mathcal{L}(n, m)$, whose models are exactly these linear hypergraphs. Before we describe the encoding itself, let us discuss the entire process including coloring and other parts on a high level. The two hallmarks of our approach are the use of *Satisfiability modulo symmetries (SMS)* [20] to enumerate hypergraphs without isomorphic copies, and an χ-edge-coloring learning technique thanks to which we do not need to enumerate all hypergraphs in order to show that they are all colorable.

▶ **Definition 6.** *Two hypergraphs $H_1 = (V, E_1)$ and $H_2 = (V, E_2)$ are* isomorphic*, written $H_1 \cong H_2$, if there is a bijection $\pi : V \to V$, extended to hyperedges and sets of hyperedges in the obvious way, so that $H_2 = \pi(H_1)$. For any total order $\preccurlyeq$ on the set of hypergraphs, we say that a hypergraph $H = (V, E)$ is* canonical *if $H \preccurlyeq \pi(H)$ for every $\pi : V \to V$.*

SMS is a general framework which augments a SAT solver with a custom propagator to check whether the currently constructed hypergraph is canonical. If not, a *symmetry clause* is learned and the solver backtracks. SMS performs *dynamic symmetry breaking*, and thus prevents the solver from redundantly exploring isomorphic (symmetric) copies of the search space. Other methods of symmetry exploitation include *static* symmetry breaking, where a single symmetry-breaking formula is added at the beginning, but those do not break all symmetries [4]. We expand on our use of SMS in Section 3.2.

The other important technique that we use is the *learning of coloring clauses*, recently proposed under the name *co-certificate learning* [18]. A naive approach to our problem, even with symmetry breaking, would consist of enumerating, modulo isomorphisms, all hypergraphs with n vertices and m hyperedges, and testing that they are all χ-edge-colorable.[1] The coloring process would come wholly after the enumeration process. We, on the other hand, interleave these two processes. As soon as we generate a candidate hypergraph, we color it. If it is not colorable, we have found a counterexample to the conjecture. If it is colorable, we learn a new clause, which we feed back to the hypergraph-generating SAT solver. From this point onwards, any generated hypergraph will not be colorable by this coloring whose clause has been learned. Thus, as soon as we discover a set of colorings that colors all canonical hypergraphs (thanks to SMS, we only generate those), the solver's active formula immediately becomes unsatisfiable. See Figure 2 for an overview of the process.

[1] Unless NP = coNP, non-colorability cannot be encoded in a polynomial-size propositional formula, i.e., we cannot simultaneously encode graph search and non-colorability.

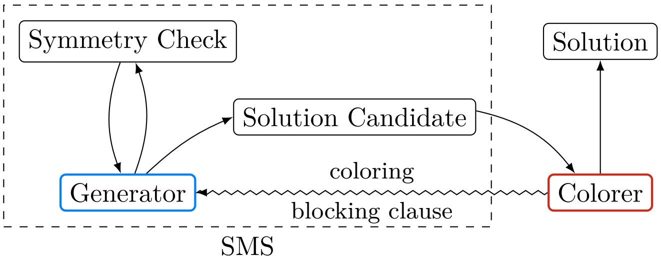


Figure 2 SMS-powered enumeration interleaved with color filtering. One SAT solver generates candidates, the other colors them, and either reports a solution, or returns a coloring clause to be learned by the generator.

We will now explain the details of each of these components: the encoding, the details of hypergraph SMS, and the coloring.

3.1 The Encoding

Our encoding of the existence of a linear hypergraphs is relatively straightforward; the magic happens in SMS and the learning of coloring clauses. We represent a hypergraph with n vertices and m hyperedges by its *incidence graph* and *incidence matrix*. The incidence graph $\mathcal{I}(H)$ of a hypergraph $H = (V, E)$ is the bipartite graph between V and E, with those edges $\{v, e\}$ where $v \in e$. The incidence matrix $\mathcal{M}(H)$ of H is the $n \times m$ binary matrix where $\mathcal{M}(H)_{i,j} = 1 \iff v_i \in e_j$, for some fixed ordering of the vertices and edges $V = \{v_1, \dots, v_n\}$, $E = \{e_1, \dots, e_m\}$.

Clearly, any of H, $\mathcal{I}(H)$, $\mathcal{M}(H)$ uniquely determines the other two, and $H \cong H' \iff \mathcal{I}(H) \cong \mathcal{I}(H')$.[2] We will use propositional variables $I_{i,j}$ to represent the incidence matrix, and we will construct incidence graphs modulo isomorphisms. As the canonical incidence graph, we pick the one with the lexicographically least incidence matrix, seen as a string of zeros and ones obtained by concatenating the rows.

We describe the encoding $\mathcal{L}(n, m)$ of the linear hypergraphs with n vertices and m edges in terms of a conjunction of clauses together with cardinality constraints. To encode cardinality constraints, we use sequential counters [24].

- To ensure all hyperedges have size at least 2, we use $\bigwedge_{j=1}^{m} \sum_{i=1}^{n} I_{i,j} \geq 2$.
- Similarly, we can request that the degree of each vertex be at least 2 (vertices of degree 1 contribute nothing to the intersection structure): $\bigwedge_{i=1}^{n} \sum_{j=1}^{m} I_{i,j} \geq 2$.
- To ensure linearity, we will use the auxiliary variables $S_{i,j,k}$ to denote that $v_k \in e_i \cap e_j$:

$$\bigwedge_{1\le i<j\le m}\ \bigwedge_{k=1}^{n} (\overline{I_{k,i}} \vee \overline{I_{k,j}} \vee S_{i,j,k}) \wedge (\overline{S_{i,j,k}} \vee I_{k,i}) \wedge (\overline{S_{i,j,k}} \vee I_{k,j}) \wedge \bigwedge_{1\le k<k'\le n} \overline{S_{i,j,k}} \vee \overline{S_{i,j,k'}}.$$

Now let us look at the constraints added to $\mathcal{L}(n, m)$ for the EFL Conjecture. We start with restricting the search to covered hypergraphs. To ensure that every pair of vertices is covered, we use the auxiliary variables $O_{k,k',i}$ to denote that $\{v_k, v_{k'}\} \subseteq e_i$:

$$\bigwedge_{1\le k<k'\le n} \left(\bigwedge_{i=1}^{m} (\overline{I_{k,i}} \vee \overline{I_{k',i}} \vee O_{k,k',i}) \wedge (\overline{O_{k,k',i}} \vee I_{k,i}) \wedge (\overline{O_{k,k',i}} \vee I_{k',i}) \right) \wedge \left(\bigvee_{i=1}^{m} O_{k,k',i} \right).$$

[2] For isomorphisms between incidence graphs we require that vertices be mapped to vertices and hyperedges to hyperedges or, what is the same, that the incidence matrix is not transposed by an isomorphism.

For the FB Conjecture we additionally parametrize the formula over the maximum size of the closed neighborhood ζ. Then $\mathcal{L}(n,m)$ is enhanced by the following two constraints:

- We use $O_{i,j}$ for $v_i, v_j \in e$ for some edge e and $\mathcal{N}(k) = \sum_{k' \in [n], k' \neq k} O_{\min(k,k'),\max(k,k')}$. To ensure that the size of the closed neighborhood is limited to ζ we add

$$\bigwedge_{1 \le k < k' \le n} \left(\bigwedge_{i \in [m]} (\overline{O_{k,k',i}} \vee O_{k,k'}) \wedge (\overline{O_{k,k'}} \vee \bigvee_{i \in [m]} O_{k,k',i}) \right) \wedge \bigwedge_{k \in [n]} \mathcal{N}(k) < \zeta;$$

- we restrict the search to weakly covered hypergraphs:

$$\bigwedge_{1 \le k < k' \le n} (O_{k,k'} \vee \mathcal{N}(k) = \zeta - 1 \vee \mathcal{N}(k') = \zeta - 1)\,.$$

  Note that the complicated-looking expression $\mathcal{N}(k) = \zeta - 1$ is a propositional variable appearing in the encoding of the cardinality constraint.

Last but not least, the variables $S_{i,j}$ to denote that $e_i \cap e_j \neq \emptyset$ are needed to describe the *intersection graph* of the hypergraph whose existence we are encoding. The intersection graph of a hypergraph $H = (V, E)$ is the graph $\mathcal{S}(H) = (E, E^{\cap})$, where $E^{\cap} = \{\{e_1, e_2\} : e_1 \neq e_2 \text{ and } e_1 \cap e_2 \neq \emptyset\}$. We will need this explicit description of the intersection graph to encode a necessary condition for a high chromatic index, and also to learn coloring clauses, as explained in Section 3.3. The variables $S_{i,j}$ can be encoded as follows: $\bigwedge_{1 \le i < j \le m} \left(\overline{S_{i,j}} \vee \bigvee_{k=1}^{n} S_{i,j,k}\right) \wedge \left(\bigwedge_{k=1}^{n} S_{i,j} \vee \overline{S_{i,j,k}}\right)$.

3.2 SMS

The encoding presented above is rich in symmetries. Any permutation of the rows and columns of the incidence matrix $\mathcal{M}(H)$ of a hypergraph H leads to an isomorphic hypergraph. Since linearity and non-colorability of the intersection graph are invariants, it is sufficient to keep only one hypergraph per isomorphism class in the search space.

Kirchweger and Szeider [20] introduced SAT modulo Symmetries (SMS) for graphs, a method that supports the search for lexicographically minimal graphs with any additional property encoded with a propositional formula. Later, it was extended to matroids of fixed rank [19]. SMS for graphs produces only lexicographically minimal graphs given by the concatenation of the rows of the adjacency matrix, i.e., those for which no permutation of the set of vertices produces a lexicographically smaller adjacency matrix. The framework checks during the CDCL procedure whether there is a permutation leading to a lexicographically smaller graph; if so, a symmetry-breaking clause is added. This is a problem with two pillars of hardness: during the search, the presence or absence of some edges might be unknown, and for those an exponential number of possibilities must be accounted for; and even if all edges are known, finding a suitable permutation is NP-hard [5]. A procedure called the *minimality check* verifies some necessary criteria for a partially defined graph to be extensible to a minimal graph. The procedure tries to construct a permutation leading to a smaller graph for all extensions of the partially defined graph. If such a permutation exists, we can add a symmetry-breaking clause to trigger a backtrack. The minimality check builds the permutation gradually using a branching algorithm for different choices. Most of the time in practice, the algorithm is fast enough, although sometimes it degrades to exponential behavior. One can limit the total number of branching steps per call of the minimality check. This potentially makes the symmetry breaking incomplete, but does not have an impact on the satisfiability of the formula. Further, it is easy to filter these copies during postprocessing using tools like Nauty [21].

Let us see how we can use SMS for the conjectures on hypergraphs. One possibility to use SMS as is, is by breaking the symmetries over the intersection graph. Preliminary tests showed that this approach does not perform well, chiefly due to the fact that for a candidate intersection graph, the solver must also find the underlying linear hypergraph, and assuming it breaks symmetries on the intersection graph, it cannot break symmetries of the hypergraph anymore. However, we can use this framework to break symmetries also for hypergraphs without (significant) changes.

Let $V = \{v_1, \ldots, v_n\}$ and the number of hyperedges of a hypergraph be fixed by m. A hypergraph H_1 is lexicographically smaller than H_2 (short $H_1 \prec H_2$) if the concatenation of the rows of the incidence matrix $\mathcal{M}(H_1)$ is lexicographically smaller than the concatenation of the rows of $\mathcal{M}(H_2)$.

Since SMS is originally designed for graphs, we apply the symmetry breaking on the incidence graph. Note that the incidence matrix $\mathcal{M}(H)$ coincides with the first n rows and last m columns of the adjacency matrix of the incidence graph $\mathcal{I}(H)$. Let us have a look at an example: let H be a hypergraph with the edges $e_1 = \{v_0, v_1, v_2\}, e_2 = \{v_0, v_3\}, e_3 = \{v_1, v_3\}$, and $e_4 = \{v_2, v_3\}$. We have

$$\mathcal{M}(H) = \begin{array}{c|cccc} & e_1 & e_2 & e_3 & e_4 \\ \hline v_0 & 1 & 1 & 0 & 0 \\ v_1 & 1 & 0 & 1 & 0 \\ v_2 & 1 & 0 & 0 & 1 \\ v_3 & 0 & 1 & 1 & 1 \end{array}, \quad \mathcal{I}(H) = \begin{array}{c|cccc|cccc|} & v_0 & v_1 & v_2 & v_3 & e_1 & e_2 & e_3 & e_4 \\ \hline v_0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ v_1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ v_2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ v_3 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ \cline{6-9} e_1 & 1 & 1 & 1 & 0 & \multicolumn{1}{c}{0} & 0 & 0 & \multicolumn{1}{c}{0} \\ e_2 & 1 & 0 & 0 & 1 & \multicolumn{1}{c}{0} & 0 & 0 & \multicolumn{1}{c}{0} \\ e_3 & 0 & 1 & 0 & 1 & \multicolumn{1}{c}{0} & 0 & 0 & \multicolumn{1}{c}{0} \\ e_4 & 0 & 0 & 1 & 1 & \multicolumn{1}{c}{0} & 0 & 0 & \multicolumn{1}{c}{0} \end{array}.$$

The red box in $\mathcal{I}(H)$ marks the part which is in common with the incidence matrix $\mathcal{M}(H)$.

▶ **Observation 7.** *$\mathcal{M}(H)$ is lexicographically minimal if and only if there is no permutation $\pi : [n+m] \to [n+m]$ with $\pi([n]) = [n]$ such that $\pi(\mathcal{I}(H)) \prec \mathcal{I}(H)$.*

SMS supports not only complete symmetry breaking but also symmetry breaking on a set of given symmetries which can be described by a *generalized ordered partition (GOP)*. Informally, a GOP gives for each vertex a range to which it can be mapped in the permutation. For a formal definition we refer to the original SMS paper [20]. In our case the set of permutations can be described as $[([n], 1, n), ([n+1, n+m], n+1, n+m)]$ which means that all elements in $[n]$ must be mapped by the permutation between 1 and n; all elements in $[n+1, n+m]$ must be mapped between $n+1$ and $n+m$.

This allows us to use SMS as follows. Let $e_{u,v}$ denote the variables representing whether the edge $\{u, v\}$ is present in the bipartite incidence graph. By definition of bipartite graph, we can add the unit clauses $\overline{e_{v,u}}$ for $v, u \in [n]$ and also $\overline{e_{v,u}}$ for $v, u \in \{n+1, \ldots, n+m\}$. Additionally, we set $e_{v,n+i} = I_{v,i}$. All clauses of $\mathcal{L}(n, m)$ can be added unmodified.

Proofs

SAT solvers, in contrast to many other tools for enumeration of combinatorial objects, can produce formal proofs, typically in the DRAT format. These proofs can be checked by external (formally verified) tools [25]. In our workflow, on top of ordinary DRAT-emitting CDCL, we add additional symmetry-breaking and coloring clauses during the search with a custom propagator altering the formula. In previous work [19] the authors suggested a multi-step approach for producing the proof. First, SMS is run until it concludes unsatisfiability.

All learned symmetry-breaking clauses are stored. Then, a second solver without SMS is run on the initial formula augmented with the symmetry-breaking clauses, and produces a proof in the ordinary fashion.

In this work, we use an adaption of the CaDiCaL solver [8] with support for custom propagators and proof logging, including in combination with propagators. The clauses added by the propagator are assumed to be part of the initial formula for the DRAT proof. Proof production in the current CaDiCaL version is only experimental.

The DRAT proof is a certificate for the correctness of the reasoning of the solver, but not for the correctness of the symmetry-breaking clauses. One can either see the symmetry-breaking clauses as part of the initial formula and apply no additional check, or one can check whether the symmetry breaking clauses follow a certain structure, and therefore preserve lexicographically minimal objects, and by extension also satisfiability. Given the permutation used for generating the symmetry-breaking clause, it is easy to check the correctness of the clause [19]. For each symmetry-breaking clause we additionally store the permutation and use a second script for checking the correctness of these clauses.

3.3 Coloring

Suppose the SMS solver has produced a hypergraph $H = (V, E)$ with $V = [n]$. We check whether the hypergraph is χ-edge-colorable with $\chi = n$ for the EFL Conjecture and $\chi = \zeta$ for the FB Conjecture. Observe that whether H is χ-edge-colorable is the same as whether its intersection graph is χ-(vertex-)colorable. We produce another propositional formula, $\mathcal{C}_{H,\chi}$, with the variables $c_{e,i}$ for each hyperedge $e \in E$ and $i \in [\chi]$ to express that e is colored with the color i, and with the following constraints:

- each edge should have some color: $\bigwedge_{e\in E} \bigvee_{i\in[\chi]} c_{e,i}$;
- intersecting edges should have different colors:

$$\bigwedge_{\substack{e_1,e_2\in E\\ e_1\neq e_2 \wedge e_1\cap e_2\neq\emptyset}} \bigwedge_{i\in[\chi]} \overline{c_{e_1,i}} \wedge \overline{c_{e_2,i}}.$$

It is not a problem if an edge receives more than one color, as extra colors can always be dropped without violating the constraints.

We enhance this basic encoding by finding and fixing the colors of a clique in the intersection graph as follows. We find the vertex v contained in the largest number of hyperedges $\{e_1, \dots, e_r\}$ and give these hyperedges the colors $1, \dots, r$ by adding the unit clauses $c_{e_1,1}, \dots, c_{e_r,r}$. We further find any other hyperedges $\{e_{r+1}, \dots, e_{r'}\}$ that intersect with all of these, for a subset-maximal clique in the intersection graph. Since these hyperedges must all have different colors, we can break the symmetries that arise from permutations of color names by fixing the colors.

A second SAT solver then finds a coloring $c : E \to [\chi]$ (a *co-certificate*, because it certifies that the hypergraph is *not* non-χ-edge-colorable [18]), and from this we learn the following *coloring clause*, which we add to the SMS SAT solver:

$$\bigvee_{k=1}^{\chi} \bigvee_{e_i,e_j\in c^{-1}(k)} S_{i,j}.$$

The coloring clause says that at least one pair of hyperedges colored the same by c should intersect, thereby invalidating the coloring c for future hypergraphs.

3.4 Restrictions on the Intersection Graph

We can impose further restrictions on the intersection graph for both conjectures based on the fact that the intersection graph should not be χ-colorable. Any graph that is not χ-colorable must contain some vertex-critical subgraph, i.e., one which is rendered χ-colorable by the removal of any vertex, but which is not χ-colorable itself. It is easy to see that a vertex-critical graph with chromatic number $\chi + 1$ has minimum degree $\geq \chi$, because adding a vertex of degree $< \chi$ to a χ-colorable graph does not break χ-colorability.

We cannot require that the whole intersection graph itself be critical because that is incompatible with (weak) coverage, but we can select a critical subgraph of the intersection graph, using additional variables s_e indicating whether a vertex is part of the subgraph, and restrict the minimum degree for this subgraph accordingly. Without loss of generality, we can additionally request that each vertex (in the intersection graph) representing a hyperedge of size ≥ 3 is part of the critical subgraph and therefore must have degree $\geq \chi$. To see that this last restriction is sound, consider the following process. Given a non-χ-colorable intersection graph $\mathcal{S}(H)$ of some (weakly) covered linear hypergraph H:

1. find a χ-vertex-critical subgraph $G \leq \mathcal{S}(H)$;
2. remove all hyperedges of size ≥ 3 from H which are not contained (as vertices) in G to obtain H';
3. add however many 2-edges are needed to make H' (weakly) covered;
4. observe that $G \leq \mathcal{S}(H')$, and so H' is not χ-edge-colorable.

A hypergraph H whose intersection graph $\mathcal{S}(H)$ contains a subgraph containing all hyperedges of size at least 3 and having minimum degree at least δ is called *δ-reduced*. It follows that if a counterexample with n vertices and m hyperedges to either conjecture exists, a $(\chi - 1)$-reduced (weakly) covered counterexample also exists, with the same number of vertices n, and a number of hyperedges $m' \geq m$.

3.5 Conjectures

Let us summarize the encoding and how the side constraints relate to the original Conjectures 1 and 3. We will formulate modified versions of the conjectures that match our encoding and precisely state their relationship to the original conjectures.

▶ **Conjecture 8** (EFL' Conjecture). *Every $(n - 1)$-reduced linear space with n vertices and m hyperedges is n-edge-colorable.*

▶ **Proposition 9.** *If there is a counterexample to the EFL Conjecture with n vertices and m hyperedges of size at least 2, then there is a counterexample to EFL' Conjecture with n vertices and $m' \geq m$ hyperedges.*

▶ **Corollary 10.** *The EFL Conjecture and the EFL' Conjecture are equivalent.*

▶ **Conjecture 11** (FB' Conjecture). *Every $(\zeta - 1)$-reduced weakly covered linear hypergraph with n vertices, m hyperedges of size at least 2, and maximum closed neighborhood size ζ is ζ-edge-colorable.*

▶ **Proposition 12.** *If there is a counterexample to FB Conjecture with n vertices and m hyperedges of size at least 2, then there is a counterexample to the FB' Conjecture with n vertices and $m' \geq m$ hyperedges.*

▶ **Corollary 13.** *The FB Conjecture and the FB' Conjecture are equivalent.*

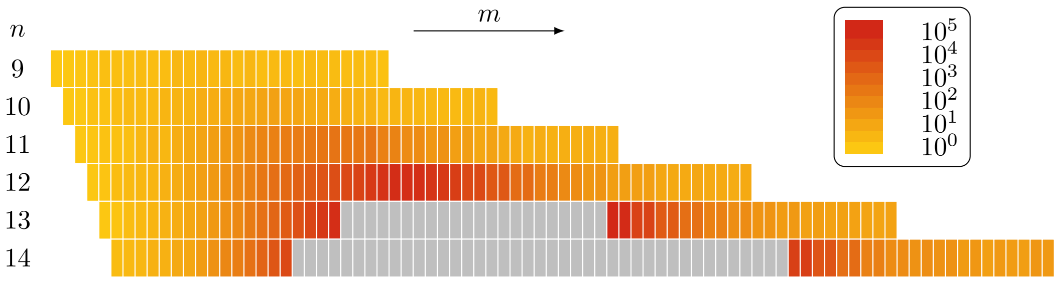


Figure 3 For selected n, the regions of m for which we verified the EFL' conjecture, within 3 days of CPU time. For each n, m runs from n to $\binom{n}{2}$. The color transitions from yellow to red in proportion to the logarithm of the running time in seconds, red means higher. White regions on the left are trivial, on the right impossible for linear spaces. Regions we could not solve are in gray.

For given n and m, the individual cases of the EFL' and FB' conjectures are, in some sense, the canonical or essential cases of EFL and FB, respectively, even though the correspondence between the (n, m)-buckets between the two versions of each conjecture is not one-to-one (and is made precise by Propositions 9 and 12).

4 Computational Results

In this section we will present the results of our computations. We performed three different experiments, each on a cluster of machines with different processors[3], running Ubuntu 18.04 on Linux 4.15. The source code and scripts for reproducing the results are available as part of the SMS package at `https://github.com/markirch/sat-modulo-symmetries`.

The first experiment was targeted at verifying Conjecture 8, and by extension the EFL conjecture itself. We managed to verify, this time with formal methods capable of proof logging, the previous known results, and we verified the conjecture in additional cases. Selected results of this experiment are summarized in Figure 3. The colored cells indicate which values of n and m we could solve, within a time limit of three days. Our results in every case agree with the prediction of Erdős, Faber, and Lovász: all linear hypergraphs are n-colorable. In addition to those shown in Figure 3, we verified all cases listed in the following theorem.

▶ **Theorem 14.** *The EFL' Conjecture (Conjecture 8) holds for n, m if $n \le 12$; or*

- $n = 13$ *and* $m \in [13, 32] \cup [55, 78]$*; or*
- $n = 14$ *and* $m \in [14, 28] \cup [70, 91]$*; or*
- $n = 15$ *and* $m \in [15, 29] \cup [84, 105]$*; or*
- $n = 16$ *and* $m \in [16, 30] \cup \{99\} \cup [101, 120]$*; or*
- $n = 17$ *and* $m \in [17, 30] \cup [117, 136]$*; or*
- $n = 18$ *and* $m \in [18, 31] \cup [134, 153]$*.*

The longest successfully terminated case that we encountered was that of $n = 15$ and $m = 85$, which took two days, 15 hours, and 26 minutes to solve, of which one day and 15 and a half hours were spent in the coloring solver.

[3] Intel Xeon {E5540, E5649, E5-2630 v2, E5-2640 v4}@ at most 2.60 GHz, AMD EPYC 7402@2.80GHz

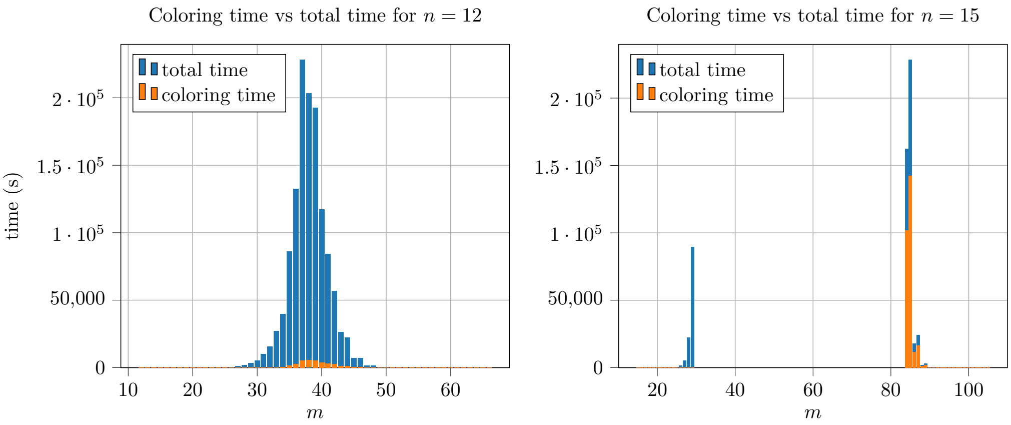


Figure 4 The distribution of the running times for $n = 12$ and $n = 15$.

In general, we observed a pattern where hardness of the individual instances culminated around the middle of the region of the available values $\left[n, \binom{n}{2}\right]$, and fell off roughly symmetrically on both sides. However, the nature of the hardness differed on the two sides: for smaller values of m, coloring was negligibly easy (less than 0.1% of the total time) and most of the time was spent in hypergraph search, while for higher values of m, coloring took up a much more significant portion of the total time; this proportion peaked for $n = 15$ and $m = 87$, at just over 67%. Selected cases are depicted in Figure 4.

For smaller values of n we can exhaustively enumerate all $(n-1)$-reduced linear spaces and compare their number with the number of learned colorings, to get an idea of the speedup. We tested this for $10 \leq n \leq 12$ and found that the total number of colorings is about 10% smaller than the number of hypergraphs. This is somewhat in contrast with much higher speedups reported in related work [18], and it would be interesting future work to investigate why our case behaves differently. One hypothesis that we have is that in our case, the intersection graphs are not canonical, and hence the colorings do not work as well. In fact, we could even see isomorphic copies of intersection graphs, which require different colorings. This is testable by setting up an appropriate experiment, but is somewhat beyond the scope of this paper.

The aim of the second experiment was to gain insights about the extremal examples to the EFL conjecture. The second experiment is similar to the first, except that

- instead of asking for hypergraphs that cannot be edge-colored with n colors (and getting none), we asked for hypergraphs that cannot be edge-colored with $n-1$ colors;
- and we dropped the constraint that the hypergraphs should be $(n-1)$-reduced.

Thus, we have enumerated all extremal linear *spaces* with respect to the EFL conjecture.

The known extremal examples are the degenerate plane (for any $n \geq 3$), the complete graph (for odd $n \geq 3$), and the projective plane, where $n = k^2 + k + 1$ for many k including all prime powers. According to Kang et al. [17], in addition to the odd clique, also "minor modifications thereof" work. Kahn [15] is similarly cryptic: "*[The EFL Conjecture] is sharp when H is a projective plane or complete graph K_n with n odd, and also in a few related cases, but there ought to be some slack in the bound away from these extremes.*" We enumerated all extremal examples with $n \leq 12$ and in some cases of $n = 13$, and can confirm that these "minor modifications," listed in Theorem 15, are the only other extremal examples.

Let $\mathcal{H}_{n,k}$ be the linear space on n vertices that consists solely of 2-edges and one k-edge, $\mathcal{H}_{n,3\not\cap 3}$ the linear space with 2-edges and two disjoint 3-edges, and $\mathcal{H}_{n,3\cap 3}$ the space with 2-edges and two intersecting 3-edges.

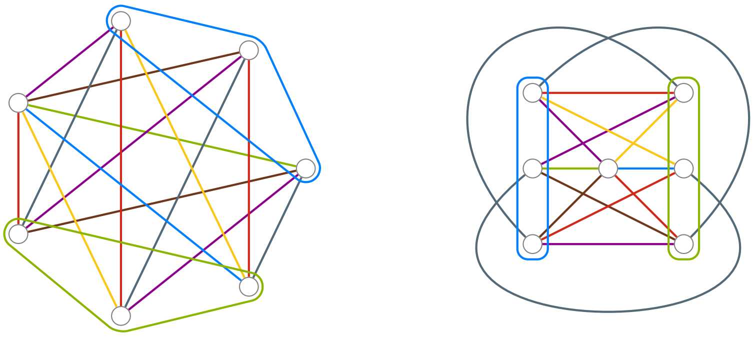


Figure 5 The extremal hypergraph $\mathcal{H}_{7,3\not\cap 3}$, on the left drawn analogously to K_7 from Figure 1, on the right in a way that highlights symmetries.

▶ **Theorem 15.** *The linear spaces with $n \leq 12$ vertices and chromatic index n are precisely $\mathcal{H}_{n,k}$ for all $k \not\equiv n \mod 2$, and additionally $\mathcal{H}_{7,3}, \mathcal{H}_{9,3}, \mathcal{H}_{11,3}$, $\mathcal{H}_{7,3\not\cap 3}$, $\mathcal{H}_{11,3\cap 3}$, $\mathcal{H}_{11,3\not\cap 3}$, and the Fano plane.*

We proved Theorem 15 almost entirely computationally, with the only exception of $\mathcal{H}_{11,3\cap 3}$ and $\mathcal{H}_{11,3\not\cap 3}$, which were too hard to prove non-colorable. In Section 4.1, we provide the missing manual proofs of extremality and generalize some patterns from Theorem 15. Theorem 15 remains necessary to show that the enumerated extremal examples are complete.

The hypergraphs $\mathcal{H}_{n,k}$ can be seen as a chain of steps of size 2 linking the complete graph to the degenerate plane. Even though the complete graph is not extremal for even n, this chain still exists, only shifted to odd k. In this case the chain can even be considered somewhat “purer,” as there is neither an “intermediate link” of the wrong parity, as there is $\mathcal{H}_{n,3}$ for odd n, nor other exceptions as listed in Theorem 15; and moreover there are no projective planes for even n ($k^2 + k + 1$ is always odd). One is almost tempted to conjecture that these are the only extremal examples when n is even.

The enumeration of extremal examples is computationally harder than the verification of the EFL conjecture, for two reasons. The first is that the coloring clauses are weaker, and as such more hypergraphs must be searched. The second is that while in the case of the EFL conjecture all graphs are colorable (all colorability queries satisfiable), in this case non-colorability sometimes has to be proved. Even though we break color symmetries by assigning arbitrary colors to one clique in the intersection graph, frequently a clique of the maximum possible size $n - 1$, a large part of the intersection graph remains to be colored, often with other large cliques. Since the formula encoding the existence of a (vertex) coloring of a clique is nothing else than the famous pigeonhole principle formula [11], known to be hard for resolution and CDCL solvers, we can expect proving non-colorability to be hard. This is indeed what we observe: for $n = 11$ we could not completely solve the case $m = 51$ ($\mathcal{H}_{11,3\cap 3}$ and $\mathcal{H}_{11,3\not\cap 3}$), and for $n = 13$ there were many cases that were easy in the first experiment, but which we could not solve now; in fact, we could not even prove (with SAT) that K_{13} is not 12-edge-colorable. Since in these cases we can easily enumerate all linear spaces, the reason for hardness must be unsatisfiable colorability queries. An interesting avenue for future research would be to improve performance on these hard coloring instances.

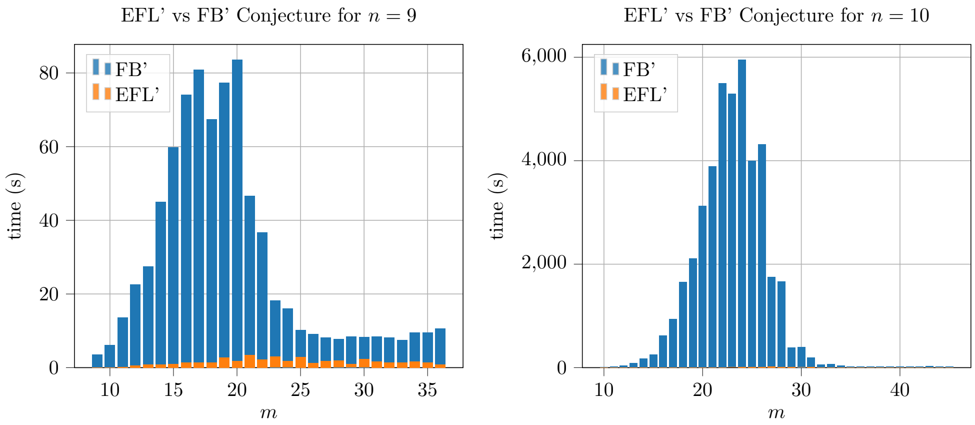


Figure 6 The distribution of the running times for the EFL' and FB' conjectures for $n \in [9, 10]$.

This experiment also uncovered that the minimality check sometimes struggles and runs out of the limit (see Section 3.2), and duplicate solutions are produced. In this case, this is not a problem in and of itself, mainly because the copies are very obviously isomorphic, but it shows that these highly symmetric hypergraphs pose a challenge for current SMS, and could serve as benchmarks for future development.

In the third experiment we tackled Conjecture 11, and through it Conjecture 3. The setup is almost identical to the first experiment, except that we use the other version of the encoding described in Section 3.1. We summarize our results in another theorem.

▶ **Theorem 16.** *The FB' Conjecture holds for n and m if $n \leq 10$; or*

- $n = 11$ *and* $m \in [11, 20] \cup [37, 55]$*; or*
- $n = 12$ *and* $m \in [12, 17] \cup [49, 66]$.

▶ **Corollary 17.** *The FB Conjecture holds for $n \leq 10$.*

Verifying the FB Conjecture is much harder than verifying the EFL conjecture; for example for $n = 11$ and $m = 35$, Conjecture 8 was solved in 12 minutes, while the same case took over a day for Conjecture 11. The running times for $n = 9, 10$ are shown in Figure 6.

4.1 Extremality Proofs

The main results of this section are Theorems 18, 22, 24, and 27. Theorem 18 says that a linear space with an odd number of vertices that does not have enough or large enough hyperedges of size ≥ 3 is extremal. Theorem 22 establishes extremality of $\mathcal{H}_{n,k}$ when n and k have opposite parity. The other two theorems fill some of the gaps left by the first two. Theorem 24 gives a sufficient condition for a space of odd size to be non-extremal, and Theorem 27 shows that spaces with an even number of vertices and only even hyperedges are non-extremal provided that large hyperedges do not intersect.

We say a color c *sees* a vertex v (and vice versa) if a c-colored hyperedge contains v.

▶ **Theorem 18.** *Let H be a linear space with n vertices, n odd, whose hyperedges of size ≥ 3 have sizes $a_1, \ldots, a_\mu$. If $n \geq \sum_{i=1}^{\mu} a_i(a_i - 2) + \mu + 2$, then H is not $(n-1)$-edge-colorable.*

Proof. For a vertex of degree d, there are $n-1-d$ colors that do not see it. We have $\deg(v) = n - 1 - \sum_{v\in e, |e|\geq 3}(|e|-2)$, so at most

$$\sum_{v\in V}\left(\sum_{v\in e,|e|\geq 3}(|e|-2)\right) = \sum_{i=1}^{\mu} a_i(a_i-2)$$

colors do not see every vertex. There are $\leq \mu$ colors used for the hyperedges of size ≥ 3, so at least one color sees every vertex and is used only on 2-edges; a contradiction for odd n. ◀

▶ **Corollary 19.** *$\mathcal{H}_{n,3\not\cap 3}$ and $\mathcal{H}_{n,3\cap 3}$ are not $(n-1)$-colorable when $n \geq 11$ is odd.*

▶ **Theorem 20.** *$\mathcal{H}_{7,3\not\cap 3}$ is not 6-edge-colorable.*

Proof. Let A and B be the 3-edges, and v the other vertex. A and B cannot be colored the same because v needs to see all colors, so the nine 2-edges between A and B use 4 colors, and some c is used ≥ 3 times. But then each vertex of A and B sees c, and so v cannot see c. ◀

▶ **Theorem 21.** *$\mathcal{H}_{9,3\not\cap 3}$ is 8-edge-colorable.*

Proof. $\{u,v\} \mapsto u+v \mod 8$ if $u,v < 8$, $\{u,8\} \mapsto 2u$ if $u \leq 3$, else $\{u,8\} \mapsto 2u-1 \mod 8$, $\{3,4,5\} \mapsto 0$, $\{6,7,8\} \mapsto 5$. ◀

▶ **Theorem 22.** *Let $n \not\equiv k \mod 2$, $k \in [2,n]$. Then $\mathcal{H}_{n,k}$ is not $(n-1)$-edge-colorable.*

Proof. Let the k-edge have the color c. All other vertices must see every color, in particular c. The color c forms a perfect matching with $1+(n-k)/2$ edges; but $n-k$ is odd. ◀

▶ **Corollary 23.** *$\mathcal{H}_{n,3}$ is not $(n-1)$-edge-colorable when $n=4$ or $n \geq 6$.*

Proof. For even n this follows from Theorem 22, for odd n from Theorem 18. ◀

▶ **Theorem 24.** *Let H be a linear space with n vertices, n odd, and assume that H contains a vertex v^* that belongs to every hyperedge of size greater than two. If the number of even hyperedges (including 2-edges) containing v^* is $\leq \frac{n-1}{2}$, then H is $(n-1)$-edge-colorable.*

Proof. Rename vertices so that $v^* = n-1$, the even hyperedges containing v^* are $A_0, \ldots, A_r$, with $i \in A_i$, and the odd hyperedges are $B_1, \ldots, B_s$, and they are ordered as follows:

$$0, 1, \ldots, r, (A_0 \setminus \{0, n-1\}), \ldots, (A_r \setminus \{r, n-1\}), (B_1 \setminus \{n-1\}), \ldots, (B_s \setminus \{n-1\}), n-1$$

Color as follows: $u, v < n-1 \implies \{u,v\} \mapsto u+v \mod n-1$, if $u \leq \frac{n-1}{2}$ and $\{u, n-1\} \subseteq E$, then $E \mapsto 2u$; all thus unassigned hyperedges must be B_i, and we color them $B_i \mapsto \min(B_i) + \max(B_i \setminus \{n-1\}) \mod n-1$. The coloring is proper because (1) the doubling color is applied to at most $\frac{n-1}{2}$ edges; (2) $\min(B_i)$ and $\max(B_i \setminus \{n-1\})$ always have opposite parity, and hence do not conflict with the doubled colors, which are even; and (3) two colors assigned by the last rule cannot be the same because $\min(B_i) \geq \frac{n+1}{2}$ $\left(\frac{n-1}{2} < a < b < c < d < n-1 \implies a+b \not\equiv c+d \mod n-1\right)$. ◀

▶ **Corollary 25.** *Let $n \leq 2k-1$, $n \equiv k \equiv 1 \mod 2$. Then $\mathcal{H}_{n,k}$ is $(n-1)$-edge-colorable.*

▶ **Corollary 26.** *$\mathcal{H}_{5,3}$ is 4-, $\mathcal{H}_{7,3\cap 3}$ is 6-, and $\mathcal{H}_{9,3\cap 3}$ is 8-edge-colorable.*

▶ **Theorem 27.** *Let H be a hypergraph with n vertices, n even. If all hyperedges of H are even, and all those of size greater than 2 are pairwise disjoint, then H is $(n-1)$-edge-colorable.*

Proof. Let the hyperedges of size greater than two be $L_1, \ldots, L_r$, let $L := \bigcup_{i=1}^{r} L_i$. Rename the vertices so that all L_i are closed under additive inverse: $x \in L_i \implies n-1-x \in L_i$. Color $L_i \mapsto 0$, $u, v < n-1 \implies \{u,v\} \mapsto u+v \mod n-1$, $\{u, n-1\} \mapsto 2u \mod n-1$. ◀

▶ **Corollary 28.** *Let $n \equiv k \equiv 0 \mod 2$, $k \in [2,n]$. Then $\mathcal{H}_{n,k}$ is $(n-1)$-edge-colorable.*

5 Conclusion

In this paper, we used SAT solvers to partially confirm two conjectures about hypergraph edge colorings for small instances. We achieved this by interleaving the two SAT solvers: one for symmetry-breaking search for hypergraphs, the other for finding colorings of candidate hypergraphs, to prune the search space early.

Our experiments show that the hypergraph search and edge-coloring problem is a challenging benchmark for all components of our framework. To improve the minimality check in SMS, we plan to combine dynamic symmetry breaking with (partial) static symmetry breaking, for example by way of sorting the vertices by degree (by a static, poly-size encoding), and limit the dynamic symmetry breaking to vertices of the same degree. Similarly, it would be very interesting to improve CDCL performance on these hard coloring problems. Here, it seems, symmetry-breaking could help, but we have a chicken-and-egg problem: if symmetry-breaking is sometimes hard for the minimality check, will it be feasible for the coloring part? In any case, hypergraphs and colorings are very general and expressive combinatorial objects, and as such are worthy of the attention to improve the performance of general-purpose SAT solvers.

References

1 Claude Berge. On the chromatic index of a linear hypergraph and the Chvátal conjecture. *Annals of the New York Academy of Sciences*, 555(1):40–44, 1989.

2 Anton Betten and Dieter Betten. Linear spaces with at most 12 points. *Journal of Combinatorial Designs*, 7(2):119–145, 1999.

3 William I. Chang and Eugene L. Lawler. Edge coloring of hypergraphs and a conjecture of Erdős, Faber, Lovász. *Combinatorica*, 8(3):293–295, 1988.

4 Michael Codish, Alice Miller, Patrick Prosser, and Peter J. Stuckey. Constraints for symmetry breaking in graph representation. *Constraints*, 24(1):1–24, 2019. doi:10.1007/s10601-018-9294-5.

5 James M. Crawford, Matthew L. Ginsberg, Eugene M. Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Stuart C. Shapiro Luigia Carlucci Aiello, Jon Doyle, editor, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996*, pages 148–159. Morgan Kaufmann, 1996.

6 Paul Erdős. Problems and results in graph theory and combinatorial analysis. *Proc. British Combinatorial Conj., 5th*, pages 169–192, 1975.

7 Pául Erdős. On the combinatorial problems which I would most like to see solved. *Combinatorica*, 1(1):25–42, 1981.

8 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. IPASIR-UP: User propagators for CDCL. In Meena Mahajan and Friedrich Slivovsky, editors, *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04–08, 2023, Alghero, Italy*, LIPIcs, pages 8:1–8:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SAT.2023.8.

9 Johannes K. Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r)evolution of SAT. *Communications of the ACM*, 66(6):64–72, June 2023. doi:10.1145/3560469.

10 Zoltán Füredi. The chromatic index of simple hypergraphs. *Graphs and Combinatorics*, 2(1):89–92, 1986.

11 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

12 Neil Hindman. On a conjecture of Erdős, Faber, and Lovász about n-colorings. *Canadian Journal of Mathematics*, 33(3):563–570, 1981. doi:10.4153/CJM-1981-046-9.

13 Jeff Kahn. Coloring nearly-disjoint hypergraphs with $n + o(n)$ colors. *Journal of combinatorial theory, Series A*, 59(1):31–39, 1992.

14 Jeff Kahn. Asymptotics of hypergraph matching, covering and coloring problems. In *Proceedings of the International Congress of Mathematicians: August 3–11, 1994 Zürich, Switzerland*, pages 1353–1362. Springer, 1995.

15 Jeff Kahn. On some hypergraph problems of Paul Erdős and the asymptotics of matchings, covers and colorings. *The Mathematics of Paul Erdős I*, pages 345–371, 1997.

16 Jeff Kahn and Paul D. Seymour. A fractional version of the Erdős-Faber-Lovász conjecture. *Combinatorica*, 12(2):155–160, 1992.

17 Dong Yeap Kang, Tom Kelly, Daniela Kühn, Abhishek Methuku, and Deryk Osthus. A proof of the Erdős-Faber-Lovász conjecture, 2021. doi:10.48550/arXiv.2101.04698.

18 Markus Kirchweger, Tomáš Peitl, and Stefan Szeider. Co-certificate learning with SAT modulo symmetries. In *Proceedings of the 34th International Joint Conference on Artificial Intelligence, IJCAI 2023*. AAAI Press/IJCAI, 2023. To appear.

19 Markus Kirchweger, Manfred Scheucher, and Stefan Szeider. A SAT attack on Rota's Basis Conjecture. In *Theory and Applications of Satisfiability Testing – SAT 2022 – 25th International Conference, Haifa, Israel, August 2-5, 2022, Proceedings*, 2022. doi:10.4230/LIPIcs.SAT.2022.4.

20 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, LIPIcs, pages 39:1–39:17. Dagstuhl, 2021. doi:10.4230/LIPIcs.CP.2021.34.

21 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symbolic Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

22 David Romero and Federico Alonso-Pecina. The Erdős–Faber–Lovász conjecture is true for $n \leq 12$. *Discrete Mathematics, Algorithms and Applications*, 06(03):1450039, 2014. doi:10.1142/S1793830914500396.

23 Paul D. Seymour. Packing nearly-disjoint sets. *Combinatorica*, 2:91–97, 1982.

24 Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming – CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer Verlag, 2005. doi:10.1007/11564751_73.

25 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer Verlag, 2014. doi:10.1007/978-3-319-09284-3_31.

SAT-Based Generation of Planar Graphs

Markus Kirchweger
Algorithms and Complexity Group, TU Wien, Austria

Manfred Scheucher
Institut für Mathematik, Technische Universität Berlin, Germany

Stefan Szeider
Algorithms and Complexity Group, TU Wien, Austria

Abstract

To test a graph's planarity in SAT-based graph generation we develop SAT encodings with dynamic symmetry breaking as facilitated in the SAT modulo Symmetry (SMS) framework. We implement and compare encodings based on three planarity criteria. In particular, we consider two eager encodings utilizing order-based and universal-set-based planarity criteria, and a lazy encoding based on Kuratowski's theorem. The performance and scalability of these encodings are compared on two prominent problems from combinatorics: the computation of planar Turán numbers and the Earth-Moon problem. We further showcase the power of SMS equipped with a planarity encoding by verifying and extending several integer sequences from the Online Encyclopedia of Integer Sequences (OEIS) related to planar graph enumeration. Furthermore, we extend the SMS framework to directed graphs which might be of independent interest.

2012 ACM Subject Classification Mathematics of computing → Solvers; Mathematics of computing → Graph enumeration; Theory of computation → Automated reasoning; Hardware → Theorem proving and SAT solving

Keywords and phrases SAT modulo Symmetry (SMS), dynamic symmetry breaking, planarity test, universal point set, order dimension, Schnyder's theorem, Kuratowski's theorem, Turán's theorem, Earth-Moon problem

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.14

Supplementary Material *Software (Source Code)*: `https://github.com/markirch/sat-modulo-symmetries/`, archived at `swh:1:dir:c50f1ce7d22717e113b915ae9976da0cb0e67c1d`
Software (Documentation): `https://sat-modulo-symmetries.readthedocs.io/`

Funding M.K. and S.S. were supported by the Austrian Science Fund (FWF), project P32441, and from the Vienna Science and Technology Fund (WWTF), project ICT19-065. M.S. was supported by DFG Grant SCHE 2214/1-1.

1 Introduction

Graph generation is the problem of deciding whether a graph with a particular property exists. Many difficult problems in Combinatorics can be stated as graph generation problems. Over the last years, SAT-based approaches to graph generation have been proposed, yielding competitive alternatives to isomorphism-free exhaustive enumeration by canonical construction path, as implemented in tools like Nauty [34]. By combining the desired graph property with symmetry breaking, SAT-based approaches can avoid generating a prohibitively large number of candidate graphs for which the desired property needs to be checked. *SAT Modulo Symmetry (SMS)* [30] is a SAT-based approach that supports complete symmetry breaking performed by a special propagator that collaborates with a CDCL SAT solver [21].

In this article, we look into SAT-based graph generation where the given property entails the graph being planar.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 14; pp. 14:1–14:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There are mainly two options for incorporating planarity into SAT-based graph search: (i) employing an "eager" encoding of planarity directly into a SAT formula or (ii) using a "lazy" encoding that incrementally adds clauses to ensure that the partially defined graph, as represented by the current partial assignment of the solver, is planar. Today, many criteria for planarity are known. Criteria that are *positive* in the sense that they state the existence of a planar embedding, are natural candidates for an eager SAT encoding because valid variable assignments are in correspondence with embeddings. Criteria that are *negative* in the sense that they state the existence of an obstruction against a planar embedding, are candidates for lazy encoding; once an obstruction has been found, the solver can exclude it by learning a corresponding clause.

In Section 3, we propose a lazy encoding based on forbidden graph minors and Kuratowski's theorem, and two eager encodings of planarity; one is based on Schnyder orders [40], and the other is based on universal point sets [13].

In Section 5, we compare the performance and scalability of these encodings on three problem settings.

The first problem setting is from extremal combinatorics [5] and seeks the maximum number of edges in graphs on n vertices that excludes a certain subgraph. The *Turán number* $\mathrm{ex}(n,H)$ for an integer n and a graph H is the maximum number of edges in an n-vertex graph G with no copy of H as a subgraph. Turán famously showed that $\mathrm{ex}(n,C_k) \leq (1-\frac{1}{k-1})\frac{n^2}{2}$, where C_k denotes the cycle graph on k vertices ([44], see also [2, Chapter 27]). Dowden [14] studied the problem restricted to planar graphs G which gives rise to the *planar Turán number* $\mathrm{ex}_P(n,H)$. Very recently, planar Turán numbers for various graphs H have become the subject of intensive research in combinatorics [12, 15, 16, 24, 32]. With our SAT-based framework, we compute planar Turán numbers for $n \leq 18$ where the excluded subgraph is a cycle of length 4 or 5, as studied in [14].

The second problem setting is an extension of the planar map coloring problem, known as the *Earth-Moon problem* introduced by Ringel [37]. It seeks the chromatic number of a biplanar graph (a graph that can be formed as the union of two planar graphs). The name of the problem originates from the figurative statement of the problem, which asks for the minimum number of colors needed to properly color a map consisting of two separate spherical (planar) maps, an Earth map containing a collection of countries, and a Moon map containing a colony for each country on Earth. A proper coloring assigns the same color to a country and its lunar colony and different colors to countries and colonies with a common boundary. It is known that the number of required colors lies between 9 and 12 (cf. [27, p. 36] and [26, p. 199]). To encode biplanar graphs for the Earth-Moon problem, we extended the SMS framework from graphs to *directed* graphs: antiparallel edges indicate edges in one planar graph and the remaining edges indicate edges in the other. This extension might be of independent interest. With this approach, we are able to show the absence of biplanar graphs with certain order and at least a certain chromatic number.

As the third problem setting, we consider various *integer sequences* related to planar graph and digraph enumeration as listed in the On-Line Encyclopedia of Integer Sequences [35]. While existing graph enumeration tools such as plantri [8] mainly aim on *plane* graphs (i.e., planar graphs with a fixed embedding), our approach is the first for planar graphs (no embeddings involved). For several of the sequences we could verify and extend the known initial segments with a relatively minor effort. In particular, having common parameters implemented (such as bounds on degrees, clique/independence number, connectivity, etc.), a large variety of sequences can be simply tested from the command line just by combining the desired parameters. We see this as an indication that, in several cases, our approach is superior and easier to use than standard graph enumeration based approaches and for the versatility of our framework.

Related Work

Chimani, Hedtke and Wiedera [10] investigated the problem of finding a planar subgraph of a given graph with the maximum number of edges. They used encodings to integer linear programs and pseudo-boolean satisfiability based on various planarity criteria for that purpose. This problem setting is very different from ours since they work with a given input graph while we aim to generate graphs for which symmetry breaking plays a central role.

Plantri is the standard tool for the generation of certain types planar graphs and was developed by Brinkmann and McKay [8]. It enumerates non-isomorphic planar graphs with a fixed embedding (*plane* graphs). Since 3-connected planar graphs have a unique embedding, plantri can directly enumerate various subclasses of 3-connected planar graphs. However, in general planar graphs can have multiple (up to exponentially many) embeddings and therefore one must filter duplicates caused by distinct embeddings. With SMS, we can enumerate planar graphs of any connectivity directly.

2 Preliminaries

For positive integers n, we write $[n] := \{1, \dots, n\}$.

We use standard notation for *CNF formulas* (propositional formulas in conjunctive normal form), propositional variables, literals, and clauses [36].

We use standard notation for graphs and digraphs [6, 45], in particular, all considered graphs and digraphs are finite and simple. A *graph* G consists of a finite set $V(G)$ of *vertices* and a set $E(G) \subseteq \{\{u, v\} : u \neq v \in V(G)\}$ of *edges*. Similarly, a *directed graph* G (or *digraph*) G consists of a finite set $V(G)$ of *vertices* and a set $E(G) \subseteq \{(u, v) : u \neq v \in V(G)\}$ of *directed edges* or *arcs*. The *underlying graph* $\underline{G}$ of a digraph G has the vertex set $\underline{G}(V) := G(V)$ and edge set $E(\underline{G}) := \{\{u, v\} : (u, v) \in E(G)\}$.

An *edge-subdivision* operation deletes an edge $\{u, v\}$ from a graph G, and adds two new edges $\{u, w\}, \{w, v\}$ and a new vertex $w \notin V(G)$. A graph G is a *subdivision* of another graph H if G can be obtained from H by successively performing edge-subdivisions.

A graph G is *connected* if there exists a path between any two vertices $u, v \in V(G)$, that is, there exists a sequence of vertices $u = w_0, w_1, \dots, w_k = v$ with $\{w_i, w_{i+1}\} \in E(G)$. Moreover, G is k*-connected* if $|V(G)| \geq k + 1$ and the deletion of any $k - 1$ vertices results in a connected graph. The *connectivity* $\kappa(G)$ denotes the largest integer k for which G is k-connected. Pause to note that the terms "k-connected" and "connectivity k" must not be confused as the class of k-connected graphs consists all graphs G with connectivity $\kappa(G) \geq k$. A digraph is *weakly* k*-connected* if its underlying graph is k-connected.

To define planarity, some auxiliary terminology is required. A *simple curve* in the plane (resp. on the sphere) is the image of a injective continuous mapping $\phi : [0, 1] \to \mathbb{R}^2$ (resp. $\phi : [0, 1] \to \mathbb{S}^2$). The points $\phi(0), \phi(1)$ are the curve's *ends* and the remaining points of the curve form the curve's *interior*. A graph is *planar* if there exists a mapping of the vertex to the plane (resp. to the sphere) and a mapping of each edge to a simple curve connecting the two corresponding vertices such the interiors of any two curves is disjoint. Such a mapping is called *embedding*. In general, one does not distinguish between embedding in the plane and embedding on the sphere since any embedding on the sphere can be transferred into the plane via a stereographic projection, and vice versa.

A *plane* graph is a planar graph with a fixed embedding. If a graph is 3-connected then it has a combinatorially unique embedding on the sphere [46], that is, the cyclic order of the incident edges around any vertex coincide in every embedding. However, graphs that are not 3-connected can have multiple embeddings, hence one planar graph can correspond to several plane graphs; see e.g. Figure 1.

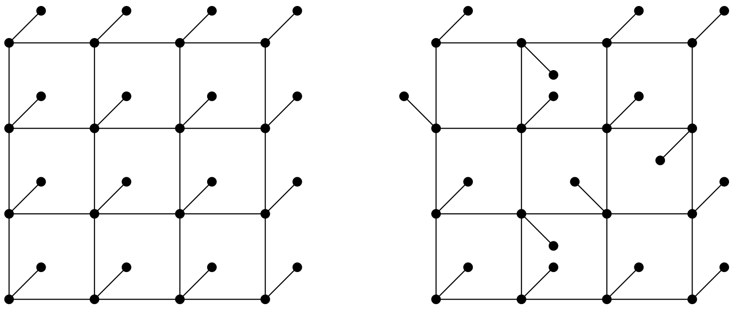


Figure 1 Two embeddings of the same planar graph. The graph is constructed by adding an edge to every vertex of the grid graph. Since every edge can be drawn in multiple cells, the graph corresponds to exponentially many non-isomorphic plane graphs.

A *k-coloring* of a graph G is mapping $c : V(G) \to [k]$ such that for every edge $\{u, v\} \in E(G)$ it holds $c(u) \neq c(v)$. A graph G is *k-colorable* if there exists a k-coloring of G and the *chromatic number* $\chi(G)$ of G denotes the smallest integer k such that G is k-colorable. The famous four-color theorem states that if G is planar then $\chi(G) \leq 4$ [3, 38].

For a graph G and permutation of the vertices $\pi : V(G) \to V(G)$, we denote the relabeled graph by $\pi(G)$, that is, $V(\pi(G)) = V(G)$ and $E(\pi(G)) = \{\{\pi(u), \pi(v)\} : \{u, v\} \in E(G)\}$.

During SAT-based graph generation, we encounter partially defined graphs and digraphs. In a *partially defined* (di)graph G, the edge set $E(G)$ is partitioned into the set $D(G)$ of *defined edges* and the set $U(G)$ of *undefined edges*. The (di)graph G is *fully defined* if $U(G) = \emptyset$. A partially defined (di)graph G can be *extended* to a fully defined (di)graph G' if $V(G') = V(G)$ and $D(G) \subseteq E(G') \subseteq D(G) \cup U(G)$. If not stated otherwise, graphs are undirected and fully defined.

3 SAT Encodings for Planarity

There are many different criteria in the literature for a graph being planar. In this section, we select three of them and implement and benchmark these encodings.

In the context of SAT-based graph generation and enumeration, the graph is not know during search, so we design the planarity encoding based on the variables describing the combinatorial object. In other words, we don't construct formulas for a given input graph, but rather for all graphs implicitly described by certain propositional variables. For a fixed number of vertices n, we use the propositional variables $e_{u,v}$, whose truth values indicate whether the edge $\{u, v\}$ is present.

3.1 Encoding Based on Kuratowski's Theorem

The famous theorem by Kuratowski asserts that a graph is planar if and only if it does not contain $K_{3,3}$ or K_5 as a topological minor, which means it does not contain a subdivision of the complete bipartite graph $K_{3,3}$ or the complete graph K_5 as a subgraph. This planarity criterion is *negative* in the sense that it is based on the non-existence of a certain object, and hence is not well suited for an eager SAT encoding.

Towards a lazy SAT encoding, note that the existence of a topological $K_{3,3}$ or K_5 minor can be checked in linear time [7, 47]. Thus we can efficiently test whether a partially defined graph can be extended to a planar graph. We can carry out such a test during the CDCL procedure, whenever an edge variable has been decided, similarly to the SMS minimality check [30]. Whenever we determine that the current partially defined graph cannot be extended to a planar graph, we add a clause preventing that the search on this partially defined graph with possible further edges continues.

For that we proceed as follows. First, we construct a partially defined graph G given by the partial assignment of the propositional edge variables. For the sake of planarity testing, we consider the fully defined graph G' with $V(G') := V(G)$, $E(G') := D(G)$, and $U(G') := \emptyset$. Since G' is a subgraph of all extensions of G, its non-planarity implies the non-planarity for all extensions of G. We apply the Boyer-Myrvold planarity testing algorithm [7] to G', a linear time planarity algorithm based on edge additions to compute a planar embedding. If it concludes that the graph G' is not planar the algorithm returns a subgraph H of G', which is a subdivision of $K_{3,3}$ or K_5. Adding the clause

$$\bigvee_{\{u,v\}\in E(H)} \neg e_{u,v}$$

blocks this specific subgraph.

3.2 Encoding Based on Schnyder Orders

Schnyder [40] proved that a graph G is planar if and only if its incidence order dimension is at most 3. Formally, there exist three partial orders $\prec_1, \prec_2, \prec_3$ (which we call *Schnyder orders*) such that for every edge $\{u,v\} \in E(G)$ and every vertex $w \in V(G) \setminus \{u,v\}$ there is some $i \in \{1,2,3\}$ such that $u \prec_i w$ and $v \prec_i w$. Since every partial order can be extended to a total order, one can assume without loss of generality that $\prec_1, \prec_2, \prec_3$ are total orders. We refer the interested reader to Chapter 2 of Felsner's book [18].

This results in a compact encoding for planarity. To enumerate all planar graphs on a vertex set $V = [n]$, we use variables $o_{u,v,i}$ to indicate whether $u \prec_i v$ and introduce the following constraints:

To ensure that $\prec_i$ is transitive, antisymmetric, and a total order, we require for $i \in \{1,2,3\}$ the following constraints.

$$\bigwedge_{\substack{u,v,w\in V\\ u\neq v\neq w\neq u}} \neg o_{u,v,i} \vee \neg o_{v,w,i} \vee o_{u,w,i}, \qquad \bigwedge_{\substack{u,v\in V\\ u\neq v}} \neg o_{u,v,i} \vee \neg o_{v,u,i}, \qquad \bigwedge_{\substack{u,v\in V\\ u\neq v}} o_{u,v,i} \vee o_{v,u,i}.$$

To ensure that $\prec_1, \prec_2, \prec_3$ form three Schnyder orders of the desired graph, we require

$$\bigwedge_{\substack{u,v\in V\\ u\neq v}} \Big(e_{u,v} \to \bigwedge_{w\in V\setminus\{u,v\}} \bigvee_{i\in\{1,2,3\}} (o_{u,w,i} \wedge o_{v,w,i})\Big).$$

The formula is transformed in a CNF formula using the Tseitin transformation [43]. This leads to $O(n^3)$ variables and $O(n^3)$ clauses.

Solutions of the SAT encoding are in correspondence with planar graphs together with a witnessing triple of orders. Pause to note that, in contrast to the Kuratowski based encoding where non-planarity is witnessed, planarity is witnessed in this encoding.

One disadvantage of this encoding is that it is not propagating, i.e., if all variables $e_{u,v}$ are assigned and the graph is not planar then Boolean constraint propagation does not necessarily lead to a conflict. Further, for a given planar graph there are at least exponentially many different witnessing triples of orders $\prec_1, \prec_2, \prec_3$ [18].

3.3 Encoding Based on Universal Sets

A set S of points from the plane is *n-universal* if every planar n-vertex graph can be embedded such that vertices are mapped S as vertices and all edges are straight-line segments. For instance, a triangular subset of the $(n-1) \times (n-1)$ grid is n-universal [41], and there exist n-universal sets of size $\frac{1}{4}n^2 - O(n)$ [4]. In general, the existence of an n-universal set of subquadratic size remains one of the central open problems of graph drawing. However, n-universal sets of minimum size have been computed for $n \leq 11$ [9, 39] and for certain subclasses of planar graphs universal sets of subquadratic size exist [19].

We want to enumerate all planar graphs G with vertex set $V = [n]$ by testing whether the graph represented by edge variables embeds into a prescribed n-universal point set S of size $k = |S|$. Note that, since all edges are drawn as straight-line segments, the injective mapping $P : V \to S$ fully determines the embedding. We use variables $m_{v,p}$ to indicate whether $P(v) = p$ and use clauses to ensure that no two edges cross. To keep the number of constraints small, we introduce auxiliary variables $s_{p,q}$ for any distinct $p, q \in S$ to indicate whether the segment determined by p and q is present. Finally, we must forbid the presence of crossing segments, i.e., segments are only allowed to share a common endpoint. We can express these conditions by the following constraints: To ensure that P is a mapping and that the relation is injective, we require

$$\bigwedge_{v \in V} \bigvee_{u \in S} m_{v,u} \quad \text{and} \quad \bigwedge_{v_1, v_2 \in V,\ u \in S} \neg m_{v_1,u} \vee \neg m_{v_2,u}.$$

To determine the presence of certain segments, we require

$$\bigwedge_{u,v \in V,\ a,b \in S} (e_{u,v} \wedge m_{u,a} \wedge m_{v,b}) \to s_{a,b}.$$

Finally, for any $a, b, a', b' \in S$ such that the segments ab and $a'b'$ intersect in a non-shared endpoint, we require

$$\neg s_{a,b} \vee \neg s_{a',b'}.$$

The intersecting segments can be precomputed based on the point set S and don't have to be determined by the SAT encoding.

For the injective mapping, we use $O(n^2 \cdot k)$ clauses, for the presence of certain segments $O(n^2 \cdot k^2)$ clauses, and for avoiding intersecting segments we use up to $O(k^4)$ clauses, where $k = |S|$ is at least n. Hence, the encoding has $O(k^4)$ clauses and $O(k^2)$ variables. A variant of this encoding was already used in [39, Section 4.3] to find universal point sets for a prescribed list of graphs.

Using the currently best n-universal point set, which are of magnitude $O(n^2)$, this encoding renders itself useless even for relatively small n due to $O(k^4) = O(n^8)$ clauses. However, we will test this approach for $n \leq 11$ since for this cases there exist reasonably sized n-universal sets.

4 SAT Modulo Symmetries and Digraphs

In this section, we describe the basic ideas of SMS [30] and how we adapt it to digraphs.

SMS is a dynamic symmetry breaking method for excluding isomorphic copies of graphs during search. It is designed to keep canonical graphs in the search space and discard all non-canonical graphs by adding symmetry breaking clauses. The canonical version is given by the lexicographically minimal adjacency matrix among all relabelings of the graphs. More precisely, a graph G is canonical if the row wise concatenation of the adjacency matrix of G is either equal or lexicographically smaller than the adjacency matrix of any relabeling $\pi(G)$.

To add symmetry breaking clauses during search, we need to be able to decide whether the partially defined graph given by the current solver state can be extended to a canonical fully defined graph. For that a minimality check was designed which checks for some necessary conditions. More precisely, it checks whether there is a permutation such that $\pi(G') \prec G'$ for all extensions of the current partial defined graph, i.e., the graph can definitely not be extended to a lexicographically minimal graph. Such a permutation is called *witness*. If the minimality check finds a witness then a symmetry breaking clause based on the current assignment and the witness permutation is constructed. The clause holds for all lexicographically minimal graphs and therefore does not exclude any potential solutions. The construction of potential witness permutations by the minimality check is based on a branching algorithm by gradually building a permutation starting with the vertex of smallest index. It is crucial for good performance to have arguments for cutting of a branch early if it does not lead to a witness permutation.

To adapt SMS for digraphs, note that all definitions for graphs used in the original SMS article [30] can be adapted to digraphs in a straight forward way. We highlight some of the adaptions in the following.

Let us start with defining a total order on the set of all digraphs $\mathcal{D}_n$ with vertex set $[n]$ for a fixed n. For that we first define an order on vertex pairs, naturally leading to an order of the digraphs. A vertex pair (v_1, v_2) is smaller than (u_1, u_2) (short $(v_1, v_2) \prec (u_1, u_2)$) if (i) $\min(v_1, v_2) < \min(u_1, u_2)$ or (ii) $\min(v_1, v_2) = \min(u_1, u_2)$ and $\max(v_1, v_2) < \max(u_1, u_2)$ or (iii) $\{v_1, v_2\} = \{u_1, u_2\}$ and $v_1 < u_1$. For example, for $n = 5$ we look at the following order of the non-diagonal elements of the $n \times n$ adjacency matrix:

$$\begin{array}{ccccc}
- & 1 & 2 & 3 & 4 \\
5 & - & 9 & 10 & 11 \\
6 & 12 & - & 15 & 16 \\
7 & 13 & 17 & - & 19 \\
8 & 14 & 18 & 20 & -
\end{array}$$

The lexicographic order on $\mathcal{D}_n$ is given by comparing the string resulting from concatenating the entries in the adjacency matrix in the order given by $\preceq$. We use $G \prec H$ for denoting that G is lexicographically smaller than H. A digraph G is $\preceq$-minimal if $G \preceq \pi(G)$ holds for all relabelings.

As in the setting of undirected graphs, the minimality check for digraphs searches for witnessing permutations. The main difference is that, while in the undirected case the adjacency matrix is symmetric and only the lower triangular matrix has to be considered, in the directed case the entire adjacency matrix needs to be checked. However, the main idea of the algorithm is the same.

A formalism presented in previous work [29] based on object variables and object symmetries guaranties that adding symmetry breaking clauses with certain structure based on some permutations of the variables does preserve lexicographically minimal objects.

5 Experiments

We test our planarity encodings in three problem settings: planar Turán numbers, the Earth-Moon problem, and planar graph enumeration. To allow comparisons between the different encodings for each problem setting separately we ensure that the programs run on the same hardware. For all encodings we use Python scripts for the generation of the clauses and feed it to our SMS framework. The underlying SAT solver is an adaption of CaDiCal with the new interface IPASIR-UP that allows the solver to interact with a custom propagator [17]; this replaces the clingo solver used previously for SMS. For the Boyer-Myrvold planarity testing algorithm [7], we use the implementation provided in the C++ Boost libraries [1].

We have developed a Python layer over SMS to ease its usage and provide better readable code (SMS is written in C++ for performance reasons). In this Python layer, we have implemented various fundamental properties and invariants for graphs and digraphs such as the bounds on the connectivity, clique number, independence number, or degrees. In particular, we have implemented the planarity encodings based on Schnyder orders and universal sets in the Python layer (see Sections 3.2 and 3.3). The Kuratowski encoding, however, is implemented in C++.

With this Python layer, it should be reasonably easy also for non-programmers to run experiments from the command line and to add additional properties for graphs and digraphs. The source code and documentation is available at GitHub[1] and Read the Docs[2], respectively.

As preliminary results show, the encoding based on universal point performs much worse than the others (see Table 5). Also recall that n-universal sets of optimal size are hard to find in general and only have been computed for $n \leq 11$. Because of these two major drawbacks, we omitted this encoding on further benchmarks.

5.1 Planar Turán Numbers

Recall that the *planar Turán number* $\mathrm{ex}_P(n, H)$ for a graph H is the maximum number of edges in a planar n-vertex graph G with no copy of H as a subgraph. We are interested in planar Turán numbers $\mathrm{ex}_P(n, C_k)$, where C_k denotes the cycle graph of length k. The case $k = 3$ is rather straight-forward: since triangle-free graphs have at most $2n - 4$ edges and $K_{2,n-2}$ obtains this bound, it holds $\mathrm{ex}_P(n, C_3) = 2n - 4$ [14]. However, the situation for $k \geq 4$ get more complicated. The currently best estimates for $k \in \{4, 5\}$ are by Dowden [14], who proved the upper bounds $\mathrm{ex}_P(n, C_4) \leq \frac{15}{7}(n - 2)$ for $n \geq 4$ and $\mathrm{ex}_P(n, C_5) \leq \frac{12n-33}{5}$ for $n \geq 11$. These bounds are tight for infinitely many values of n. For example, for $k = 4$ the bound is tight for all $n \equiv 30 \pmod{70}$, and for $k = 5$ it is tight for all $n \equiv 9 \pmod{15}$.

Using our planarity encodings, we determine the exact values of $\mathrm{ex}_P(n, C_4)$ and $\mathrm{ex}_P(n, C_5)$ for small values of n. We construct a formula $F_{n,m,k}$ which is satisfiable if there is a C_k-free graph with at least m edges. To encode C_k-free graphs we explicitly, we add the clause

$$\neg e_{v_1,v_2} \vee \neg e_{v_2,v_3} \vee \cdots \vee \neg e_{v_{k-1},v_k} \vee \neg e_{v_k,v_1}$$

for any k distinct vertices $v_1, \ldots, v_k \in V$. Using sequential counters [42], we ensure that the number of edges is at least m.

Given the ideas in Section 3 for ensuring planarity of the generated graphs, we compute the exact values of $\mathrm{ex}_P(n, C_k)$. For a fixed n and k, this is done by testing $F_{n,m,k}$ enhanced with a planarity encoding for satisfiability, starting with $m = n$. We increment m until the formula is unsatisfiable. Our computational result are summarized by the following theorem.

[1] https://github.com/markirch/sat-modulo-symmetries/

[2] https://sat-modulo-symmetries.readthedocs.io/

Table 1 Result for computing $\mathrm{ex}_P(n, C_4)$. All computation times are given in seconds. The third column gives the upper bound by Dowden [14]. SMS also found a graph with 19 vertices and 35 edges within 14 seconds, but we are not aware if this example is extremal for $n = 19$.

			SAT		UNSAT	
n	$\mathrm{ex}_P(n, C_4)$	$\lfloor \frac{15}{7}(n-2) \rfloor$	Kura	Ord	Kura	Ord
4	4	4	0.00	0.00	0.00	0.00
5	6	6	0.00	0.00	0.00	0.00
6	7	8	0.00	0.00	0.00	0.01
7	9	10	0.01	0.01	0.01	0.02
8	11	12	0.01	0.02	0.02	0.03
9	13	15	0.03	0.04	0.05	0.05
10	16	17	0.04	0.07	0.07	0.06
11	18	19	0.16	0.44	0.16	0.23
12	20	21	0.27	0.98	0.56	2.29
13	22	23	0.23	0.14	1.96	15.27
14	24	25	0.20	0.44	6.46	340.11
15	27	27	1.00	0.85	21.39	294.07
16	29	30	5.87	24.90	172.90	31142.08
17	31	32	5.19	83.59	3479.65	t.o.
18	33	34	14.69	14.85	59862.72	t.o.

▶ **Theorem 1.** *It holds that*

$$\mathrm{ex}_P(n, C_4) = \left\lfloor \frac{15}{7}(n-2) \right\rfloor - \begin{cases} 0 & \text{for } n \in \{4, 5, 15\}, \\ 1 & \text{for } n \in [6, 8] \cup [10, 14] \cup [16, 18], \\ 2 & \text{for } n = 9, \end{cases}$$

and

$$\mathrm{ex}_P(n, C_5) = \left\lfloor \frac{12n - 33}{5} \right\rfloor + \begin{cases} 0 & \text{for } n \in \{9\} \cup [11, 18], \\ 1 & \text{for } n \in \{8, 10\}, \\ 2 & \text{for } n \in [5, 7]. \end{cases}$$

Moreover, based on our computational data, we conjecture that Dowden's upper bound for $k = 5$ is tight for all $n \geq 11$.

▶ **Conjecture 2.** $\mathrm{ex}_P(n, C_5) = \lfloor \frac{12n-33}{5} \rfloor$ *for* $n \geq 11$.

Tables 1 and 2 summarize the computation times for both encodings. The times for solving $F_{n,\mathrm{ex}_P(n,C_k),k}$ are given by "SAT" and $F_{n,\mathrm{ex}_P(n,C_k)+1,k}$ given by "UNSAT". Computations not finished within three days are marked with "t.o." (timeout). The columns labeled "Kura" provide the times for the encoding based on Kuratowski's theorem with a propagator and the columns "Ord" provides the times for the encoding based on Schnyder orders.

In general, we see that the version excluding Kuratowski graphs performs much better, especially for unsatisfiable cases. For example for $n = 17, k = 4$ the Kuratowski based generation is over a hundred times faster than the encoding based on Schnyder orders.

Table 2 Result for computing $\mathrm{ex}_P(n, C_5)$. All computation times are given in seconds. The third column gives the upper bound by Dowden [14] for $n \geq 11$.

			SAT		UNSAT	
n	$\mathrm{ex}_P(n, C_5)$	$\lfloor \frac{12n-33}{5} \rfloor$	Kura	Ord	Kura	Ord
5	7	5	0.00	0.01	0.00	0.00
6	9	7	0.00	0.01	0.01	0.01
7	12	10	0.01	0.02	0.02	0.01
8	13	12	0.02	0.07	0.05	0.11
9	15	15	0.03	0.06	0.07	0.38
10	18	17	0.10	0.29	0.23	1.67
11	19	19	0.12	0.30	0.57	4.89
12	22	22	1.83	1.72	1.99	33.08
13	24	24	0.48	1.61	11.45	271.18
14	27	27	3.18	7.63	35.24	1174.85
15	29	29	2.24	10.82	277.78	15459.24
16	31	31	4.71	59.09	3172.27	235353.58
17	34	34	207.49	890.98	29023.55	t.o.
18	36	36	1851.84	1249.38	t.o.	t.o.

5.2 The Earth-Moon Problem

A graph G is *biplanar* if it can be partitioned into two planar graphs, that is, there exist two planar graphs G_1, G_2 with $E(G) = E(G_1) \cup E(G_2)$. In that case, we write $G = G_1 \uplus G_2$. Biplanar graphs are also known as graphs with *thickness* two. The Earth-Moon problem asks for the largest chromatic number a biplanar graph can have, denoted by χ_2. In 1973, Thom Sulanke constructed a biplanar graph on 11 vertices with chromatic number 9 by removing the edges of a C_5 from a K_{11}, improving an earlier lower bound by Ringel to $\chi_2 \geq 9$ [22]. On the other hand, using Euler's formula, one can derive that any biplanar graph must have a vertex of degree at most 11, which applied inductively shows that $\chi_2 \leq 12$. Despite of much research efforts, the estimates $9 \leq \chi_2 \leq 12$ could not be improved since then. Some have suggested that this problem is "as hard as two or three four-color theorems" [26, p. 199].

Searching for biplanar graphs and at least a certain chromatic number seems to be an extremely challenging problem. Indeed, the problem of deciding whether a graph is biplanar is NP-complete [33] and checking whether a graph has at least chromatic number χ for a fixed constant $\chi \geq 3$ is coNP-complete in general [28]. To admit partial progress, one can parameterized the Earth-Moon problem by the number n of vertices in the biplanar graph, denoting the highest chromatic number for a n-vertex biplanar graph by $\chi_2(n)$. Sulanke's lower bound $\chi_2(11) \geq 9$ carries over to $n > 11$ since adding isolated vertices to a biplanar graph does not change its chromatic number and keeps the graph biplanar.

Our goal is to show the absence or presence of biplanar graphs for given order n and chromatic number χ using SMS and planarity encodings.

One possibility of using SMS for biplanar graphs is applying the symmetry breaking directly at the graph G. This way, we would take edge variables $e_{u,v}$ describing the graph G. To encode the decomposition $G = G_1 \uplus G_2$, we introduce auxiliary variables $e^1_{u,v}$ and $e^2_{u,v}$ to indicate whether an edge $\{u, v\}$ belongs to $E(G_1)$ or $E(G_2)$, respectively. However, this way we don't break all symmetries. If π is an automorphism of G, i.e., $\pi(G) = G$, then it does not necessarily hold that $\pi(G_1) = G_1$ and $\pi(G_2) = G_2$. In other words, we will get different partitions representing isomorphic decompositions. This is a real problem in practice as some experiments on testing biplanarity of K_9 showed.

Hence, we propose a different and more efficient approach. Instead of encoding the biplanar graph G directly, we represent the decomposition $G_1 \uplus G_2$ as a directed graph H with $\underline{H} = G$. H represents the decomposition $G_1 \uplus G_2$ as follows.

- $\{u, v\} \in E(G_1)$ if and only if $(u, v) \in E(H)$ and $(v, u) \in E(H)$.
- $\{u, v\} \in E(G_2)$ if and only if either $(u, v) \in E(H)$ or $(v, u) \in E(H)$, but not both.

Now we can apply SMS for digraphs as discussed in Section 4. Consider two directed graphs H and H' that represent the decompositions $G_1 \uplus G_2 = \underline{H}$ and $G'_1 \uplus G'_2 = \underline{H'}$, respectively. We observe that if H and H' are isomorphic, then $\underline{H}$ and $\underline{H'}$ are isomorphic and G_i and G'_i are isomorphic, $i \in \{1, 2\}$. Consequently, it is sound to only consider lexicographically minimal digraphs H.

We further restrict the digraphs. W.l.o.g., we may assume that if $(u, v) \in E(H)$ for $u < v$ then also $(v, u) \in E(H)$. This is the case because $(u, v) \prec (v, u)$, hence replacing the arc (u, v) by (v, u) if (v, u) is not present leads to a strictly lexicographically smaller graph representing the same decomposition.

We note that the symmetry breaking on biplanar graphs using digraphs still has some potential room for improvement. There are non-isomorphic digraphs H, H' whose underlying graphs $\underline{H}, \underline{H'}$ are isomorphic, i.e., we have different representation for the same underlying graph. For example, if the underlying graph is $\underline{H} = K_5$ (the complete graph on 5 vertices), we can partition the graph $\underline{H} = G_1 \uplus G_2$ in almost all ways granted that both G_1 and G_2 contain at least one edge and none contains all edges. Further, the representation as digraphs doesn't exclude all isomorphic partitions, i.e., there are lexicographically minimal digraphs with the described restrictions representing isomorphic partitions. We plan to design a version of SMS avoiding these isomorphic copies in the future.

W.l.o.g., we may assume for a decomposition $G = G_1 \uplus G_2$ that G_1 is *maximal planar*, i.e., inserting any additional edge makes the graph non-planar, since we can move as many edges as possible from G_2 to G_1. We encode this by requiring that $|E(G_1)| = 3n - 6$, hence we can also require $|E(G_2)| \leq 3n - 6$.

Further, we restrict our search on *vertex-critical* graphs with respect to the chromatic number χ, i.e., deleting any vertex decreases the chromatic number of G. Hence we can assume that the minimum degree of G is $\geq \chi - 1$.

The following encoding describes the digraph H with vertex set $V = [n]$ that represents the decomposition of a χ-chromatic graph $\underline{H} = G$ into two planar subgraphs. We use directed edge variables $d_{u,v}$ to encode the existence of the directed edge $(u, v) \in E(H)$.

To restrict the digraph, we require

$$\bigwedge_{\substack{v,u \in V\\ v<u}} d_{v,u} \to d_{u,v} \qquad \text{and} \qquad \bigwedge_{\substack{v,u \in V\\ v<u}} e^1_{v,u} \leftrightarrow (d_{v,u} \wedge d_{u,v});$$

this results in $e^1_{v,u} \leftrightarrow d_{v,u}$ for $v < u$. We further require

$$\bigwedge_{\substack{v,u \in V\\ v<u}} e^2_{v,u} \leftrightarrow (\neg d_{v,u} \wedge d_{u,v}) \qquad \text{and} \qquad e_{v,u} = e^1_{v,u} \wedge e^2_{v,u},$$

which can be simplified to $e_{v,u} \leftrightarrow d_{u,v}$ for $v < u$. Finally, we require

$$\sum_{\substack{v,u \in V\\ v<u}} e^1_{v,u} = 3n - 6 \qquad \text{and} \qquad \sum_{\substack{v,u \in V\\ v<u}} e^2_{v,u} \leq 3n - 6,$$

encoded with sequential counters [42].

Table 3 Computations for the Earth-Moon problem, given the number n of vertices and the chromatic number χ. If the timeout of 2 days is reached we write “t.o.”.

	$\chi \geq 9$			$\chi \geq 10$		
n	#digraph	Kura	Ord	#digraph	Kura	Ord
9	0	0.55	18.14			
10	0	15.75	2028.21	0	1.95	112.43
11	5554	1709.49	t.o.	0	19.03	4543.13
12	-	t.o.	t.o.	0	837.14	t.o.
13	-	t.o.	t.o.	0	146484.00	t.o.

For ensuring at least a certain chromatic number χ, we add *coloring clauses* ensuring that the underlying graph cannot be colored with $\chi - 1$ colors. Let $\mathcal{P}_n$ be the set of all partitions of V. Then

$$\bigwedge_{\substack{P\in\mathcal{P}_n\\|P|=\chi-1}} \bigvee_{S\in P} \bigvee_{\substack{u,v\in V\\u<v}} e_{u,v}$$

ensures that every $(\chi-1)$-coloring is no proper coloring of the underlying graph for $\chi - 1 \geq n$, because at least one edge is monochromatic. Since the number of partitions $\mathcal{P}_n$ is exponential, this size of the encoding grows exponentially. However, as our experiments showed, this approach is still feasible for small values of n. We have also tried a lazy encoding which adds the clauses incrementally whenever there is a violation instead of adding all clauses right at the beginning. As it turned out, the results for this version were worse and hence we omit the results for the lazy version.

Table 3 shows the results and computation times of our experiments. For $\chi \geq 9$ and $n = 11$ the formula is satisfiable and the previously known results were confirmed. For $\chi \geq 10$ and $n \leq 13$ the formula is unsatisfiable. Therefore we have the following result.

▶ **Theorem 3.** *All biplanar graphs on $n \leq 13$ vertices are 9-colorable.*

Our experiments again show that the Kuratowski-based encoding is superior by orders of magnitudes. Table 4 summarizes the new results in context of what has been known so far.

In the literature, there are some potential candidates for the Earth-Moon Problem, which are known to have chromatic number 10, but haven't been shown to be biplanar yet [23]. One of these graphs is $G = C_5[4,4,4,4,3]$, i.e., a 5-cycle where the first four vertices of the cycle are inflated to a 4-clique, and the last to a 3-clique. The graph has 19 vertices and 99 edges. We can test whether this graph is biplanar using our planarity encodings. This can be done by adding constraints that ensure that the underlying graph of the resulting directed graph is the graph G:

$$\bigwedge_{u,v\in E(G),u<v} e_{v,u} \wedge \bigwedge_{u,v\in V(G),u,v\notin E(G)} \neg e_{v,u}.$$

By fixing some of the directed edges, SMS is not applicable anymore for all permutations. We only allow permuting vertices within the 4-clique and 3-clique, respectively, which preserves the underlying graph G. Within 12 hours, we are able to show that the graph is not biplanar, hence we can exclude the graph as a potential candidate.

Table 4 Current state of knowledge on the Earth-Moon problem for n-vertex biplanar graphs for $8 \leq n \leq 18$. Orange cells indicate that there doesn't exist an n-vertex biplanar graph with chromatic number χ, blue cells indicate the existence. The cells labeled "new" correspond to new results obtained in this paper, using the observations that removing an independent set decreases the chromatic number by at most 1 and since K_n with $n \geq 9$ is not biplanar, every biplanar graph with $n \geq 9$ vertices has an independent set of size two. With a minimality argument it is also possible to exclude the case with $n = 18$ and $\chi = 12$. If $n < \chi$, the problem is trivially unsatisfiable. If $\chi = n$, then the only potential n-vertex graph is the complete graph K_n; for $n \leq 8$, K_n is known to be biplanar, for $n \geq 9$ it is not biplanar. All biplanar graphs are known to have a chromatic number ≤ 12, hence all cells in the rightmost column are orange. The cases $n \geq 11$ and $\chi = 9$ are all satisfiable, as witnessed by Sulanke's graph.

	chromatic number χ					
n	8	9	10	11	12	13
8	K_8	🟧	🟧	🟧	🟧	🟧
9	🟦	K_9	🟧	🟧	🟧	🟧
10	🟦	new	K_{10}	🟧	🟧	🟧
11	🟦	Sulanke	new	K_{11}	🟧	🟧
12	🟦	🟦	new	new	K_{12}	🟧
13	🟦	🟦	new	new	new	K_{13}
14	🟦	🟦	open	new	new	🟧
15	🟦	🟦	open	new	new	🟧
16	🟦	🟦	open	open	new	🟧
17	🟦	🟦	open	open	new	🟧
18	🟦	🟦	open	open	new	🟧
19	🟦	🟦	open	open	open	🟧

5.3 Integer Sequences Related to Planar Graphs and Digraphs

Many integer sequences featured in the *On-Line Encyclopedia of Integer Sequences (OEIS)* [35] give the number of non-isomorphic n-vertex graphs with a certain property, for $n \in \mathbb{N}$. The encyclopedia is very useful for research in combinatorics because a sequence can for instance be used to come up with a hypothetical closed formula for a sequence, or to check whether two graph classes coincide. Often, no closed formula for a sequence is known; therefore, only a finite prefix is reported on OEIS. In this section, we demonstrate the versatility of our SMS framework in conjunction with the new planarity encoding to almost effortlessly verify and extend sequences listed on OEIS. Moreover, it allows us to compute and add new natural sequences for which no suitable tools have existed.

In the following, we review some specific integer sequences that we could verify or extend with SMS. The list is not exhaustive and can certainly be improved by further optimization.

Let us start with the sequence for non-isomorphic planar n-vertex graphs OEIS/A5470; the precise numbers are known for up to $n = 12$. Table 5 shows the running times required to verify these numbers with SMS and the three planarity encodings. Since the encoding based on Kuratowski's theorem performs significantly better than the other two, we only used this encoding in the following.

Table 5 Enumeration of planar graphs with SMS.

n	# (OEIS/A5470)	Kura	Ord	Univ
2	2			
3	4			
4	11			
5	33			
6	142	0.02s	0.04s	0.02s
7	822	0.12s	0.27s	0.24s
8	6966	1.02s	3.48s	6.94s
9	79853	13.5s	1m16s	8m50s
10	1140916	5m56s	2h16m	116h
11	18681008	13h53m		
12	333312451			

OEIS/A49339 counts the number of n-vertex planar graphs with even degrees. With previous tools, the first 12 terms were computed (Brendan McKay gave 11). We verified these 12 terms with SMS and extended the sequence by the 13th and 14th terms (about 2 and 40 hours of computation time, respectively).

OEIS/A49339 is also the Euler transform of OEIS/A49365, which counts the number of *connected* n-vertex planar graphs with even degrees. Therefore, having n terms of one sequence, one can compute the n terms of the other. Surprisingly, SMS performed almost twice as fast for computing the 13th and 14th term on OEIS/A49339.

The sequences OEIS/A49369 to OEIS/A49373 count the number of planar graphs with minimum degree at least $k \in \{1, 2, 3, 4, 5\}$. Verifying all terms for $k = 3, 4, 5$ using SMS took about 3 hours, 1 hour, and 2 days, respectively. Moreover, we have extended OEIS/A49372 (the sequence for $k = 4$) by the 16th term, which was computed within 2 days, and OEIS/A49373 (the sequence for $k = 5$) by the 26th term, which was computed within 8 days,

OEIS/A255600 counts the number of connected planar regular graphs on $2n$ vertices with a girth of at least 4. Note that girth at least 4 is equivalent to C_3-free (a.k.a. *triangle-free*) and, as noted in the comments of that sequence, all such graphs are 3-regular. SMS can verify the previous 13 terms within 90 minutes. We have extended the sequence by the 14th and 15th term, for which the computations took 16 hours and 9 days, respectively.

OEIS/A58378 counts the number of 3-regular 2-connected planar $2n$-vertex graphs. SMS verified all known terms up to $n = 13$ (i.e., up to 26 vertices) within 5 days.

While plantri was used to enumerate k-connected planar graphs for up to $k = 4$, it is surprising that there was no OEIS entry yet for *5-connected planar graphs.* So we created OEIS/A361578.

There was no OEIS entry yet for *planar digraphs*, so we created it OEIS/A361366 for up to $n = 6$. Note that, when compared with the number of planar graphs, the two options for directing each edge cause an increase in the numbers exponentially. Table 6 gives an overview of k-connected graphs and weakly k-connected digraphs for $k \leq 5$ for both general and planar settings. Since planar (directed) graphs have connectivity at most 5, we here only discuss the case $k \leq 5$. For more information on higher connectivity on general graphs, we refer to the table in OEIS/A259862.

Only sequences for weakly k-connected digraphs with $k \in \{0, 1\}$ were known; hence we created sequences for $k \in \{2, 3\}$. We also created sequences for weakly k-connected planar directed graphs for all $k \in \{0, \ldots, 3\}$ and added them to OEIS. Surprisingly, when we

Table 6 Sequences for k-connected planar graphs and weakly k-connected planar digraphs. Entries marked with $*$ are new.

	graphs		digraphs	
k-connected	general	planar	general	planar
$k = 0$	OEIS/A88	OEIS/A5470	OEIS/A273	OEIS/A361366*
$k = 1$	OEIS/A1349	OEIS/A3094	OEIS/A3085	OEIS/A361368*
$k = 2$	OEIS/A2218	OEIS/A21103	OEIS/A361367*	OEIS/A361369*
$k = 3$	OEIS/A6290	OEIS/A944	OEIS/A361370*	OEIS/A361371*
$k = 4$	OEIS/A86216	OEIS/A7027	?	?
$k = 5$	OEIS/A86217	OEIS/A361578*	?	?

recently submitted our results to OEIS, Andrew Howroyd extended the sequence for weakly 2-connected graphs by using an approach based on generating functions and combinatorial species. This gives a beautiful example of how our contribution can stimulate research in enumerative combinatorics.

Last, we should mention the well-understood class of maximal planar graphs, known as triangulations. The entries OEIS/A109, OEIS/A7021, and OEIS/A111358 count 3, 4, and 5-connected triangulations, respectively.

6 Conclusion

We have presented a comprehensive study on SAT-based planar graph generation using encodings with dynamic symmetry breaking. Our experimental results compare the effectiveness and scalability of the Kuratowski-based and order-based encodings in solving combinatorial problems related to planar graphs. In particular, we provided progress concerning the computation of planar Turán numbers and the Earth-Moon problem. Furthermore, we have shown the potential of the SMS framework equipped with planarity encodings by verifying and extending several OEIS sequences related to planar graph enumeration.

Additionally, we suggest exploring the adaptation of the Kuratowski [18, Section 1.4] and Schnyder encodings [20] for outerplanar graphs, which presents an interesting application area for SMS.

For planar graphs, there exists a polynomial-time canonization algorithm [25, 31]. Cook's Theorem [11] allows us to translate this algorithm into a polynomially-sized SAT encoding for planar graph canonization. It would be interesting to see, whether such a symmetry breaking tailored to planar graphs outperforms the general symmetry breaking in a practical setting.

References

1 David Abrahams et al. Boost C++ Libraries. `http://www.boost.org/`.

2 Martin Aigner and Günter M. Ziegler. *Proofs from The Book*, chapter Chapter 41: Turán's graph theorem, pages 285–289. Springer, Berlin, sixth edition, 2018. `doi:10.1007/978-3-662-57265-8`.

3 Kenneth Appel and Wolfgang Haken. The solution of the four-color-map problem. *Sci. Amer.*, 237(4):108–121, 152, 1977. `doi:10.1038/scientificamerican1077-108`.

4 Michael J. Bannister, Zhanpeng Cheng, William E. Devanny, and David Eppstein. Superpatterns and universal point sets. *J. Graph Algorithms Appl.*, 18(2):177–209, 2014. doi:10.7155/jgaa.00318.

5 Béla Bollobás. *Extremal graph theory.* Academic Press, 1978.

6 John A. Bondy and Uppaluri S.R. Murty. *Graph Theory.* Springer Verlag, 1st edition, 2008.

7 John M. Boyer and Wendy J. Myrvold. On the Cutting Edge: Simplified $O(n)$ Planarity by Edge Addition. *J. Graph Algorithms Appl.*, 8(3):241–273, 2004. doi:10.7155/jgaa.00091.

8 Gunnar Brinkmann and Brendan D. McKay. Fast generation of planar graphs. *MATCH Communications in Mathematical and in Computer Chemistry*, 58(2):323–357, 2007.

9 Jean Cardinal, Michael Hoffmann, and Vincent Kusters. On universal point sets for planar graphs. In *Computational Geometry and Graphs*, pages 30–41. Springer, 2013. doi:10.1007/978-3-642-45281-9_3.

10 Markus Chimani, Ivo Hedtke, and Tilo Wiedera. Exact algorithms for the maximum planar subgraph problem: New models and experiments. *ACM J. Exp. Algorithmics*, 24(1):2.1:1–2.1:21, 2019. doi:10.1145/3320344.

11 Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7(4):354–375, 1973. doi:10.1016/S0022-0000(73)80029-7.

12 Daniel W. Cranston, Bernard Lidický, Xiaonan Liu, and Abhinav shantanam. Planar Turán numbers of cycles: a counterexample. *Electron. J. Combin.*, 29(3):Paper No. 3.31, 10, 2022. doi:10.37236/10774.

13 Hubert de Fraysseix, János Pach, and Richard Pollack. Small Sets Supporting Fary Embeddings of Planar Graphs. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC'88)*, pages 426–433. Association for Computing Machinery, 1988. doi:10.1145/62212.62254.

14 Chris Dowden. Extremal C_4-free/C_5-free planar graphs. *J. Graph Theory*, 83(3):213–230, 2016. doi:10.1002/jgt.21991.

15 Liangli Du, Bing Wang, and Mingqing Zhai. Planar Turán numbers on short cycles of consecutive lengths. *Bull. Iranian Math. Soc.*, 48(5):2395–2405, 2022. doi:10.1007/s41980-021-00644-1.

16 Longfei Fang, Bing Wang, and Mingqing Zhai. Planar Turán number of intersecting triangles. *Discrete Math.*, 345(5):Paper No. 112794, 10, 2022. doi:10.1016/j.disc.2021.112794.

17 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and Armin Biere. IPASIR-UP: User propagators for CDCL. In Meena Mahajan and Friedrich Slivovsky, editors, *The 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), July 04–08, 2023, Alghero, Italy*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. to appear.

18 Stefan Felsner. *Geometric Graphs and Arrangements – Some Chapters from Combinatorial Geometry.* Advanced Lectures in Mathematics (ALM). Vieweg+Teubner, 2004. doi:10.1007/978-3-322-80303-0.

19 Stefan Felsner, Hendrik Schrezenmaier, Felix Schröder, and Raphael Steiner. Linear size universal point sets for classes of planar graphs, 2023. arXiv:2303.00109, to appear in Proceedings of the 39th International Symposium on Computational Geometry (SoCG 2023).

20 Stefan Felsner and William T. Trotter. Posets and planar graphs. *Journal of Graph Theory*, 49(4):273–284, 2005. doi:10.1002/jgt.20081.

21 Johannes K. Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r)evolution of SAT. *Communications of the ACM*, 66(6):64–72, June 2023. doi:10.1145/3560469.

22 Martin Gardner. Mathematical games. *Sci. Amer.*, 242(1):22–33B, 1980.

23 Ellen Gethner. *To the Moon and Beyond*, pages 115–133. Springer Verlag, 2018. doi:10.1007/978-3-319-97686-0_11.

24 Debarun Ghosh, Ervin Győri, Ryan R. Martin, Addisu Paulos, and Chuanqi Xiao. Planar Turán number of the 6-cycle. *SIAM J. Discrete Math.*, 36(3):2028–2050, 2022. doi:10.1137/21M140657X.

25 J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 172–184. Association for Computing Machinery, 1974. doi:10.1145/800119.803896.

26 Joan P. Hutchinson. Some conjectures and questions in chromatic topological graph theory. In *Graph theory—favorite conjectures and open problems. 1*, Probl. Books in Math., pages 195–210. Springer, 2016. doi:10.1007/978-3-319-31940-7_12.

27 Tommy R. Jensen and Bjarne Toft. *Graph coloring problems.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., New York, 1995. A Wiley-Interscience Publication.

28 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103. Plenum, New York, 1972.

29 Markus Kirchweger, Manferd Scheucher, and Stefan Szeider. A SAT attack on Rota's basis conjecture. In *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022*, volume 236 of *LIPIcs*, pages 4:1–4:18. Schloss Dagstuhl, 2022. doi:10.4230/LIPIcs.SAT.2022.4.

30 Markus Kirchweger and Stefan Szeider. SAT modulo symmetries for graph generation. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, LIPIcs, pages 39:1–39:17. Dagstuhl, 2021. doi:10.4230/LIPIcs.CP.2021.34.

31 Jacek P. Kukluk, Lawrence B. Holder, and Diane J. Cook. Algorithm and experiments in testing planar graphs for isomorphism. *Journal of Graph Algorithms and Applications*, 8(3):313–356, 2004. doi:10.7155/jgaa.00094.

32 Yongxin Lan, Yongtang Shi, and Zi-Xia Song. Extremal H-free planar graphs. *Electron. J. Combin.*, 26(2):Paper No. 2.11, 17, 2019. doi:10.37236/8255.

33 Anthony Mansfield. Determining the thickness of graphs is NP-hard. *Math. Proc. Cambridge Philos. Soc.*, 93(1):9–23, 1983. doi:10.1017/S030500410006028X.

34 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symbolic Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

35 OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences. Published electronically at http://oeis.org.

36 Steven D. Prestwich. CNF encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 75–100. IOS Press, 2021. doi:10.3233/FAIA200985.

37 Gerhard Ringel. *Färbungsprobleme auf Flächen und Graphen*, volume 2 of *Mathematische Monographien.* Berlin: VEB Deutscher Verlag der Wissenschaften, 1959.

38 Neil Robertson, Daniel Sanders, Paul Seymour, and Robin Thomas. The four-colour theorem. *J. Combin. Theory Ser. B*, 70(1):2–44, 1997. doi:10.1006/jctb.1997.1750.

39 Manfred Scheucher, Hendrik Schrezenmaier, and Raphael Steiner. A note on universal point sets for planar graphs. *J. Graph Algorithms Appl.*, 24(3):247–267, 2020. doi:10.7155/jgaa.00529.

40 Walter Schnyder. Planar graphs and poset dimension. *Order*, 5:323–343, 1989. doi:10.1007/BF00353652.

41 Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 138–148. Society for Industrial and Applied Mathematics, 1990. doi:10.5555/320176.320191.

42 Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *11th International Conference on Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer Verlag, 2005.

43 Grigori S. Tseitin. Complexity of a derivation in the propositional calculus. *Zap. Nauchn. Sem. Leningrad Otd. Mat. Inst. Akad. Nauk SSSR*, 8:23–41, 1968. Russian. English translation in J. Siekmann and G. Wrightson (eds.) *Automation of Reasoning. Classical Papers on Computer Science 1967–1970*, Springer Verlag, 466–483, 1983.

44 Paul Turán. Eine Extremalaufgabe aus der Graphentheorie. *Középiskolai Matematikai és Fizikai Lapok*, 48:436–452, 1941.

45 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2000.

46 Hassler Whitney. Non-separable and planar graphs. *Proceedings of the National Academy of Sciences of the United States of America*, 17(2):125–127, 1931. URL: `http://www.jstor.org/stable/86196`.

47 Stanley G. Williamson. Depth-First Search and Kuratowski Subgraphs. *Journal of the ACM*, 31(4):681–693, 1984. `doi:10.1145/1634.322451`.

On CNF Conversion for Disjoint SAT Enumeration

Gabriele Masina
DISI, University of Trento, Italy

Giuseppe Spallitta
DISI, University of Trento, Italy

Roberto Sebastiani
DISI, University of Trento, Italy

Abstract

Modern SAT solvers are designed to handle problems expressed in Conjunctive Normal Form (CNF) so that non-CNF problems must be CNF-ized upfront, typically by using variants of either Tseitin or Plaisted and Greenbaum transformations. When passing from solving to enumeration, however, the capability of producing partial satisfying assignments that are as small as possible becomes crucial, which raises the question of whether such CNF encodings are also effective for enumeration.

In this paper, we investigate both theoretically and empirically the effectiveness of CNF conversions for disjoint SAT enumeration. On the negative side, we show that: (i) Tseitin transformation prevents the solver from producing short partial assignments, thus seriously affecting the effectiveness of enumeration; (ii) Plaisted and Greenbaum transformation overcomes this problem only in part. On the positive side, we show that combining Plaisted and Greenbaum transformation with NNF preprocessing upfront – which is typically not used in solving – can fully overcome the problem and can drastically reduce both the number of partial assignments and the execution time.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Hardware → Theorem proving and SAT solving; Computing methodologies → Representation of Boolean functions

Keywords and phrases CNF conversion, AllSAT, AllSMT

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.15

Related Version *Extended Version*: `https://arxiv.org/abs/2303.14971` [13]

Supplementary Material *Software (Source Code)*: `https://github.com/masinag/allsat-cnf`
archived at `swh:1:dir:9fe50303a2947f6392f1601ec3c326a9582fdfb5`

Funding We acknowledge the support of the MUR PNRR project FAIR – Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU. The work was partially supported by the project "AI@TN" funded by the Autonomous Province of Trento.

1 Introduction

State-of-the-art SAT and SMT solvers deal very efficiently with formulas expressed in Conjunctive Normal Form (CNF). In real-world scenarios, however, it is common for problems to be expressed as non-CNF formulas. Hence, these problems must be converted into CNF before being processed by the solver. This conversion is generally done by using variants of the Tseitin [21] or the Plaisted and Greenbaum [17] transformations, which generate a linear-size equisatisfiable CNF formula by labelling sub-formulas with fresh Boolean atoms. These transformations can be employed also for SAT and SMT enumeration (also referred to in the literature as AllSAT and AllSMT), by projecting the models on the original atoms only.

When passing from SAT to AllSAT, however, the capability of enumerating partial satisfying assignments that are as small as possible is crucial, because each prevents from enumerating a number of total assignments that is exponential w.r.t. the number of unassigned atoms. This raises the question of whether CNF encodings conceived for solving are also effective for enumeration. To the best of our knowledge, however, no research has yet been published to analyse how the different CNF encodings may affect the effectiveness of the solvers for AllSAT and AllSMT.

In this paper, we investigate, both theoretically and empirically, the effectiveness of CNF conversion for enumeration. We focus on AllSAT, restricting to disjoint enumeration. We expect analogous results for AllSMT. The contribution of this paper is twofold. First, on the negative side, we show that the commonly employed CNF transformations for SAT are not suitable for AllSAT. In particular, we notice that the Tseitin encoding introduces top-level label definitions for sub-formulas with double implications, which need to be satisfied as well and thus prevent the solver from producing short partial assignments. We also notice that the Plaisted and Greenbaum transformation solves this problem only in part by labelling sub-formulas only with single implications if they occur with single polarity, but it has similar issues to the Tseitin transformation when sub-formulas occur with both polarities. Second, on the positive side, we show that converting the formula into Negation Normal Form (NNF) before applying the Plaisted and Greenbaum transformation can fix the problem and drastically improve the effectiveness of the enumeration process by up to orders of magnitude.

This analysis is confirmed by an experimental evaluation of non-CNF problems originating from both synthetic and real-world-inspired applications. The results confirm the theoretical analysis, showing that the combination of NNF with the Plaisted and Greenbaum CNF allows for a significant reduction in both the number of partial assignments and the execution time.

Related Work

The impact of using different CNF encodings on the performance for SAT and SMT solving has been widely studied in the literature [3, 10, 2, 11].

Beyond the basic task of SAT and SMT solving, several applications in probabilistic reasoning require quantifying the number of solutions of a SAT or SMT formula. Whereas for some applications it is sufficient to count the number of satisfying assignments, others require to enumerate all of them. In particular, SAT and SMT disjoint enumeration play a foundational role in probabilistic reasoning frameworks such as #SMT [5] and Weighted Model Integration [14, 15, 19]. Specifically, in the case of #SMT($\mathcal{LRA}$) we need to sum up the volumes corresponding to each of the models, whereas in WMI we need to integrate some function w over the polytopes defined by each of the models. Hence, in these cases, it is essential to enumerate disjoint partial models that are as small and as few as possible.

The problem of model minimization for Tseitin-encoded problems was addressed by [9]. They first propose to simplify the formula by considering its original structure and the current model; then they use iterative calls to a SAT solver to obtain a minimal model by imposing increasingly tighter cardinality constraints. This approach can be used to find a single short model, but it can be very expensive and thus it is unsuitable for model enumeration.

Content

The paper is organized as follows. In §2 we introduce the theoretical background necessary to understand the rest of the paper. In §3 we analyse the problem of the classical CNF-izations when used for AllSAT. In §4 we propose one possible solution, whose effectiveness is evaluated on both synthetic and real-world inspired benchmarks in §5. We conclude the paper in §6, drawing some final remarks and indicating possible future work.

2 Background

This section introduces the notation and the theoretical background necessary to understand what is presented in this paper. We recall the standard syntax, semantics, and results of propositional logic, and the fundamental ideas behind SAT enumeration and projected enumeration implemented by modern AllSAT solvers.

2.1 Propositional Logic

In this section, we summarize some basic definitions and results of propositional logic.

Notation and Terminology

In the paper, we adopt the following conventions. We refer to propositional atoms with capital letters, such as A and B. Propositional formulas are referred to with Greek letters such as φ, ψ. Total truth assignments are denoted by η, while partial truth assignments are denoted by μ. The symbols $\mathbf{A} \overset{\text{def}}{=} \{A_1, \ldots, A_N\}$ and $\mathbf{B} \overset{\text{def}}{=} \{B_1, \ldots, B_K\}$ denote disjoint sets of propositional atoms. We denote Boolean constants by $\mathcal{B} \overset{\text{def}}{=} \{\top, \bot\}$.

A *propositional formula* φ can be defined recursively as follows. The Boolean constants $\top$ and $\bot$ are formulas; a Boolean atom A and its negation $\neg A$ are formulas, also referred to as *literals*; a connection of two formulas φ and ψ by one of the connectors $\wedge, \vee, \to, \leftrightarrow$ is a formula. A sub-formula occurs with *positive* [resp. *negative*] *polarity* (also *positively* [resp. *negatively*]) if it occurs under an even [resp. odd] number of nested negations. Specifically, φ occurs positively in φ; if $\neg\varphi_1$ occurs positively [resp. negatively] in φ, then φ_1 occurs negatively [resp. positively] in φ; if $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ occur positively [resp. negatively] in φ, then φ_1 and φ_2 occur positively [resp. negatively] in φ; if $\varphi_1 \to \varphi_2$ occurs positively [resp. negatively] in φ, then φ_1 occurs negatively [resp. positively] and φ_2 occurs positively [resp. negatively] in φ; if $\varphi_1 \leftrightarrow \varphi_2$ occurs in φ, then φ_1 and φ_2 occur both positively and negatively in φ.

Negation Normal Form

A Boolean formula is in *Negation Normal Form (NNF)* iff it is given only by the recursive applications of $\wedge$ and $\vee$ to literals. A formula can be converted into NNF by recursively rewriting implications $(\alpha \to \beta)$ as $(\neg\alpha \vee \beta)$ and equivalences $(\alpha \leftrightarrow \beta)$ as $(\neg\alpha \vee \beta) \wedge (\alpha \vee \neg\beta)$, and then by recursively "pushing down" the negations: $\neg(\alpha \vee \beta)$ as $(\neg\alpha \wedge \neg\beta)$, $\neg(\alpha \wedge \beta)$ as $(\neg\alpha \vee \neg\beta)$ and $\neg\neg\alpha$ as α. If the NNF formula is represented as a DAG, then its size is linear w.r.t. the original one. (Although this fact is well-known, we provide a formal proof in the extended version of this paper [13].) Intuitively, we only need at most 2 nodes for each sub-formula φ_i of φ, representing $\mathsf{NNF}(\varphi_i)$ and $\mathsf{NNF}(\neg\varphi_i)$ for positive and negative occurrences of φ_i respectively. These nodes are shared among up to exponentially-many branches generated by expanding the nested iffs.

CNF Transformations

A Boolean formula is in *Conjunctive Normal Form (CNF)* iff it is a conjunction $(\wedge)$ of clauses, where a clause is a disjunction $(\vee)$ of literals. Numerous CNF transformation procedures, commonly referred to as CNF-izations, have been proposed in the literature. In the next paragraph, we summarize the three most frequently employed techniques.

The *Classic CNF-ization* ($\mathsf{CNF_{DM}}$) converts any propositional formula into a logically equivalent formula in CNF by applying DeMorgan's rules. First, it converts the formula into NNF. Second, it recursively rewrites sub-formulas $\alpha \vee (\beta \wedge \gamma)$ as $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$ to distribute $\vee$ over $\wedge$, until the formula is in CNF. The principal limitation of this transformation lies in the possible exponential growth of the resulting formula compared to the original (e.g. when the formula is in DNF), making it unsuitable for modern SAT solvers [1].

The *Tseitin CNF-ization* ($\mathsf{CNF_{Ts}}$) [21] avoids this exponential blow-up by labelling each sub-formula φ_i with a fresh Boolean atom B_i, which is used as a placeholder for the sub-formula. Specifically, it consists in applying recursively bottom-up the rewriting rule $\varphi \Longrightarrow \varphi[\varphi_i|B_i] \wedge \mathsf{CNF_{DM}}(B_i \leftrightarrow \varphi_i)$ until the resulting formula is in CNF, where $\varphi[\varphi_i|B_i]$ is the formula obtained by substituting in φ every occurrence of φ_i with B_i.

The *Plaisted and Greenbaum CNF-ization* ($\mathsf{CNF_{PG}}$) [17] is a variant of the $\mathsf{CNF_{Ts}}$ that exploits the polarity of sub-formulas to reduce the number of clauses of the final formula. Specifically, if a sub-formula φ_i appears only with positive [resp. negative] polarity, then it can be labelled with a single implication as $\mathsf{CNF_{DM}}(B_i \to \varphi_i)$ [resp. $\mathsf{CNF_{DM}}(B_i \leftarrow \varphi_i)$].

With both $\mathsf{CNF_{Ts}}$ and $\mathsf{CNF_{PG}}$, due to the introduction of the label variables, the final formula does not preserve the equivalence with the original formula but only the equisatisfiability. Moreover, they also have a stronger property. If $\varphi(\mathbf{A})$ is a non-CNF formula and $\psi(\mathbf{A} \cup \mathbf{B})$ is either the $\mathsf{CNF_{Ts}}$ or the $\mathsf{CNF_{PG}}$ encoding of φ, where $\mathbf{B}$ are the fresh Boolean atoms introduced by the transformation, then $\varphi(\mathbf{A}) \equiv \exists \mathbf{B}.\psi(\mathbf{A} \cup \mathbf{B})$.

Total and partial truth assignments

Given a set of Boolean atoms $\mathbf{A}$, a *total truth assignment* is a total map $\eta : \mathbf{A} \longmapsto \mathcal{B}$. A *partial truth assignment* is a partial map $\mu : \mathbf{A} \longmapsto \mathcal{B}$. Notice that a partial truth assignment represents 2^K total truth assignments, where K is the number of unassigned variables by μ. With a little abuse of notation, we sometimes represent a truth assignment either as a set, s.t. $\mu \overset{\text{def}}{=} \{A \mid \mu(A) = \top\} \cup \{\neg A \mid \mu(A) = \bot\}$, or as a cube, s.t. $\mu \overset{\text{def}}{=} \bigwedge_{\mu(A)=\top} A \wedge \bigwedge_{\mu(A)=\bot} \neg A$. If $\mu_1 \subseteq \mu_2$ [resp. $\mu_1 \subset \mu_2$] we say that μ_1 is a *sub-assignment* [resp. *strict sub-assignment*] of μ_2 and that μ_2 is a *super-assignment* [resp. *strict super-assignment*] of μ_1. We denote with $\varphi|_\mu$ the *residual of φ under μ*, i.e. the formula obtained by substituting in φ each $A_i \in \mathbf{A}$ with $\mu(A_i)$, and by recursively applying the standard propagation rules of truth values through Boolean operators.

Given a set of Boolean atoms $\mathbf{A}$ and a formula $\varphi(\mathbf{A})$, we say that a *[partial or total] truth assignment $\mu : \mathbf{A} \longmapsto \mathcal{B}$ satisfies φ*, denoted as $\mu \models \varphi$, iff $\varphi|_\mu = \top$[1]. If $\mu \models \varphi$, then we say that μ is a *model* of φ. A partial truth assignment μ is *minimal* for φ iff $\mu \models \varphi$ and every strict sub-assignment $\mu' \subset \mu$ is such that $\mu' \not\models \varphi$.

Most of the modern SAT and SMT solvers do not deal directly with non-CNF formulas, rather they convert them into CNF by using either $\mathsf{CNF_{Ts}}$ or $\mathsf{CNF_{PG}}$. As seen in the previous paragraph, since these transformations introduce fresh atoms into the resulting formulas, a model of $\varphi(\mathbf{A})$ can be found as a truth assignment satisfying $\exists \mathbf{B}.\psi(\mathbf{A} \cup \mathbf{B})$. Given two disjoint sets of Boolean atoms $\mathbf{A}, \mathbf{B}$ and a CNF formula $\psi(\mathbf{A} \cup \mathbf{B})$, we say that a *[partial or total] truth assignment $\mu^{\mathbf{A}} : \mathbf{A} \longmapsto \mathcal{B}$ satisfies $\exists \mathbf{B}.\psi$* iff there exists a total truth assignment $\eta^{\mathbf{B}} : \mathbf{B} \longmapsto \mathcal{B}$ such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} : \mathbf{A} \cup \mathbf{B} \longmapsto \mathcal{B}$ satisfies ψ.

[1] The definition of satisfiability by partial assignment may present some ambiguities for non-CNF and existentially-quantified formulas [18, 16]. Here we adopt the above definition because it is the easiest to implement, and it is the one typically used by state-of-the-art SAT solvers. We refer to [18, 16] for details.

Algorithm 1 MINIMIZE-ASSIGNMENT$(\psi_i, \eta_i, \mathbf{A})$ // $\psi_i \overset{\text{def}}{=} \psi \wedge \bigwedge_{j=1}^{i-1} \neg\mu_j^{\mathbf{A}}$, $\eta_i = \eta_i^{\mathbf{A}} \cup \eta_i^{\mathbf{B}}$.

```
$\mu_i^{\mathbf{A}} \leftarrow \eta_i^{\mathbf{A}}$
for $\ell \in \mu_i^{\mathbf{A}}$ do
    if $\psi_i|_{[\mu_i^{\mathbf{A}} \setminus \{\ell\} \,\cup\, \eta_i^{\mathbf{B}}]} = \top$ then
        $\mu_i^{\mathbf{A}} \leftarrow \mu_i^{\mathbf{A}} \setminus \{\ell\}$
return $\mu_i^{\mathbf{A}}$
```

2.2 AllSAT and Projected AllSAT

AllSAT is the task of enumerating all the models of a propositional formula. In this paper, we focus on the enumeration of disjoint models, that is, pairwise mutually-inconsistent models. Given a Boolean formula φ, we denote with $\mathcal{TTA}(\varphi) \overset{\text{def}}{=} \{\eta_1, \dots, \eta_j \dots \eta_M\}$ the set of all total truth assignments satisfying φ. We denote with $\mathcal{TA}(\varphi) \overset{\text{def}}{=} \{\mu_1, \dots, \mu_i \dots, \mu_N\}$ a set of partial truth assignments satisfying φ s.t.:

(a) every $\eta \in \mathcal{TTA}(\varphi)$ is a super-assignment of some $\mu \in \mathcal{TA}(\varphi)$;

(b) every pair $\mu_i, \mu_j \in \mathcal{TA}(\varphi)$ assigns opposite truth value to at least one atom.

Notice that, whereas $\mathcal{TTA}(\varphi)$ is unique, multiple $\mathcal{TA}(\varphi)$s are admissible for the same formula φ, including $\mathcal{TTA}(\varphi)$. AllSAT is the task of enumerating either $\mathcal{TTA}(\varphi)$ or a set $\mathcal{TA}(\varphi)$. Typically, AllSAT solvers aim at enumerating a set $\mathcal{TA}(\varphi)$ as small as possible, since every partial model prevents from enumerating a number of total models that is exponential w.r.t. the number of unassigned atoms, so that to save computational space and time.

The enumeration of a $\mathcal{TA}(\varphi)$ for a non-CNF formula φ is typically implemented by first converting it into CNF, and then enumerating its models by means of *Projected AllSAT*. Specifically, let $\varphi(\mathbf{A})$ be a non-CNF formula and let $\psi(\mathbf{A} \cup \mathbf{B})$ be the result of applying either $\mathsf{CNF}_{\mathsf{Ts}}$ or $\mathsf{CNF}_{\mathsf{PG}}$ to φ, where $\mathbf{B}$ is the set of Boolean atoms introduced by either transformations. $\mathcal{TA}(\varphi)$ is enumerated via Projected AllSAT as $\mathcal{TA}(\exists \mathbf{B}.\psi)$, i.e. as a set of (partial) truth assignments over $\mathbf{A}$ that can be extended to total models of ψ over $\mathbf{A} \cup \mathbf{B}$. We refer to the general schema described in [12], which we briefly recap here.

Let $\psi(\mathbf{A} \cup \mathbf{B})$ be a CNF formula over two disjoint sets of Boolean variables $\mathbf{A}, \mathbf{B}$, where $\mathbf{A}$ is a set of *relevant atoms* s.t. we want to enumerate a $\mathcal{TA}(\exists \mathbf{B}.\psi)$. The solver enumerates one-by-one partial truth assignments $\mu_1, \dots, \mu_i, \dots \mu_N$, where each $\mu_i \overset{\text{def}}{=} \mu_i^{\mathbf{A}} \cup \eta_i^{\mathbf{B}}$ is s.t.:

(i) (*satisfiability*) $\mu_i \models \psi$;

(ii) (*disjointness*) for each $j < i$, $\mu_i^{\mathbf{A}}, \mu_j^{\mathbf{A}}$ assign opposite truth values to some atom in $\mathbf{A}$;

(iii) (*minimality*) $\mu_i^{\mathbf{A}}$ is *minimal*, meaning that no literal can be dropped from it without losing properties (i) and (ii).

A basic disjoint AllSAT procedure (implemented e.g. in MATHSAT [6]) works as follows. At each step i, it finds a total truth assignment $\eta_i \overset{\text{def}}{=} \eta_i^{\mathbf{A}} \cup \eta_i^{\mathbf{B}}$ s.t. $\eta_i \models \psi_i$, where $\psi_i \overset{\text{def}}{=} \psi \wedge \bigwedge_{j=1}^{i-1} \neg\mu_j^{\mathbf{A}}$, and then invokes a minimization procedure on $\eta_i^{\mathbf{A}}$ to compute a partial truth assignment $\mu_i^{\mathbf{A}}$ satisfying properties (i), (ii) and (iii). Then, the solver adds the clause $\neg\mu_i^{\mathbf{A}}$ to ensure property (ii) and it continues the search. This process is iterated until ψ_{N+1} is found to be unsatisfiable for some N, and the set $\{\mu_i^{\mathbf{A}}\}_{i=1}^{N}$ is returned.

The minimization procedure consists in iteratively dropping literals one-by-one from $\eta_i^{\mathbf{A}}$, checking if it still satisfies the formula. The outline of this minimization procedure is shown in Algorithm 1. Each minimization step is $O(\#\mathit{clauses} \cdot \#\mathit{vars})$.

Notice that, since we are in the context of *projected* AllSAT, the minimization algorithm only minimizes the relevant variables in $\mathbf{A}$, and the truth value of existentially quantified variables in $\mathbf{B}$ is still used to check the satisfiability of the formula by the current partial assignment. Moreover, to enforce the pairwise disjointness between the assignments, ψ_i in Algorithm 1 refers to the original formula conjoined with all current blocking clauses $\bigwedge_{j=1}^{i-1} \neg\mu_j^{\mathbf{A}}$, whereas conflict clauses are excluded by the minimization, being redundant.

We stress the fact that the work described in this paper is agnostic w.r.t. the disjoint AllSAT procedure used, provided its output assignments match conditions (i)-(iii) above.

3 The impact of CNF transformations for AllSAT

In this section we analyze the impact of different CNF-izations on the AllSAT task. In particular, we focus on $\mathsf{CNF_{Ts}}$ [21] and $\mathsf{CNF_{PG}}$ [17]. We point out how CNF-izing AllSAT problems using these transformations can introduce unexpected drawbacks in the enumeration process. In fact, we show that the resulting encodings can prevent the solver from effectively minimizing the models, and thus from enumerating a small set of short partial models.

3.1 The impact of Tseitin CNF transformation

We show that preprocessing the input formula using the $\mathsf{CNF_{Ts}}$ transformation [21] can be problematic for enumeration. We first illustrate this issue with an example.

▶ **Example 1.** Consider the propositional formula

$$\varphi \overset{\text{def}}{=} \overbrace{(A_1 \wedge A_2)}^{B_1} \vee (\overbrace{\underbrace{(\overbrace{(A_3 \vee A_4)}^{B_2} \wedge \overbrace{(A_5 \vee A_6)}^{B_3})}_{B_4} \leftrightarrow A_7)}^{B_5} \tag{1}$$

over the set of atoms $\mathbf{A} \overset{\text{def}}{=} \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$. We first notice that the minimal partial truth assignment:

$$\mu^{\mathbf{A}} \overset{\text{def}}{=} \{\neg A_3, \neg A_4, \neg A_7\} \tag{2}$$

suffices to satisfy φ, even though it does not assign a truth value to the sub-formulas $(A_1 \wedge A_2)$ and $(A_5 \vee A_6)$ since the atoms A_1, A_2, A_5, A_6 are not assigned.

Nevertheless, φ is not in CNF, and thus it must be CNF-ized by the solver before starting the enumeration process. If $\mathsf{CNF_{Ts}}$ is used, then the following CNF formula is obtained:

$$\begin{aligned}
&\mathsf{CNF_{Ts}}(\varphi) \overset{\text{def}}{=} \\
&\quad (\neg B_1 \vee A_1) \wedge (\neg B_1 \vee A_2) \wedge (B_1 \vee \neg A_1 \vee \neg A_2) \wedge && //(B_1 \leftrightarrow (A_1 \wedge A_2)) && \text{(3a)}\\
&\quad (B_2 \vee \neg A_3) \wedge (B_2 \vee \neg A_4) \wedge (\neg B_2 \vee A_3 \vee A_4) \wedge && //(B_2 \leftrightarrow (A_3 \vee A_4)) && \text{(3b)}\\
&\quad (B_3 \vee \neg A_5) \wedge (B_3 \vee \neg A_6) \wedge (\neg B_3 \vee A_5 \vee A_6) \wedge && //(B_3 \leftrightarrow (A_5 \vee A_6)) && \text{(3c)}\\
&\quad (\neg B_4 \vee B_2) \wedge (\neg B_4 \vee B_3) \wedge (B_4 \vee \neg B_2 \vee \neg B_3) \wedge && //(B_4 \leftrightarrow (B_2 \wedge B_3)) && \text{(3d)}\\
&\quad (\neg B_5 \vee B_4 \vee \neg A_7) \wedge (\neg B_5 \vee \neg B_4 \vee A_7) \wedge && //(B_5 \leftrightarrow (B_4 \leftrightarrow A_7)) && \text{(3e)}\\
&\quad (B_5 \vee B_4 \vee A_7) \wedge (B_5 \vee \neg B_4 \vee \neg A_7) \wedge && && \\
&\quad (B_1 \vee B_5) \wedge && && \text{(3f)}
\end{aligned}$$

The fresh atoms $\mathbf{B} \overset{\text{def}}{=} \{B_1, B_2, B_3, B_4, B_5\}$ label sub-formulas as in (1). The solver proceeds to compute $\mathcal{TA}(\exists\mathbf{B}.\mathsf{CNF_{Ts}}(\varphi))$ by enumerating the models of $\mathsf{CNF_{Ts}}(\varphi)$ projected over $\mathbf{A}$. Suppose, e.g., that the solver picks non-deterministic choices, deciding the atoms in the order $\{B_1, A_1, A_2, B_2, A_3, A_4, B_3, A_5, A_6, B_4, B_5, A_7\}$ and branching with negative value first. Then, the first (sorted) total truth assignment found is:

$$\eta \overset{\text{def}}{=} \{\underbrace{\neg B_1, \neg B_2, \neg B_3, \neg B_4, B_5}_{\eta^{\mathbf{B}}}, \underbrace{\neg A_1, \neg A_2, \neg A_3, \neg A_4, \neg A_5, \neg A_6, \neg A_7}_{\eta^{\mathbf{A}}}\} \tag{4}$$

which contains $\mu^{\mathbf{A}}$ (2). The minimization procedure looks for a *minimal* subset $\mu^{\mathbf{A}\prime}$ of $\eta^{\mathbf{A}}$ s.t. $\mu^{\mathbf{A}\prime} \cup \eta^{\mathbf{B}} \models \mathsf{CNF}_{\mathsf{Ts}}(\varphi)$. One possible output of this procedure is the minimal assignment:

$$\mu^{\mathbf{A}\prime} \overset{\text{def}}{=} \{\neg A_1, \neg A_3, \neg A_4, \neg A_5, \neg A_6, \neg A_7\}. \tag{5}$$

We notice that the partial truth assignment $\mu^{\mathbf{A}}$ (2) satisfies φ and it is s.t. $\mu^{\mathbf{A}} \subset \mu^{\mathbf{A}\prime}$, but it *does not satisfy* $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$. In fact, three clauses of $\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$ in (3a) and (3c) are not satisfied by $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}}$, since $\mathsf{CNF}_{\mathsf{Ts}}(\varphi)|_{\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}}} = (\neg A_1 \vee \neg A_2) \wedge (\neg A_5) \wedge (\neg A_6)$. We remark that this is not a coincidence, since there is no $\eta^{\mathbf{B}\prime}$ such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}\prime} \models \mathsf{CNF}_{\mathsf{Ts}}(\varphi)$, because (3a) and (3c) cannot be satisfied without assigning any atom in $\{A_1, A_2\}$ and $\{A_5, A_6\}$ respectively.

Finding $\mu^{\mathbf{A}\prime}$ (5) instead of $\mu^{\mathbf{A}}$ (2) clearly causes an efficiency problem, since finding longer partial models implies that the total number of enumerated models could be up to exponentially larger. For instance, instead of the single partial assignment $\mu^{\mathbf{A}}$ (2), the solver may return the following list of 9 partial assignments satisfying $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$:

$$\begin{array}{lccccccccl}
 & \multicolumn{2}{c}{\overbrace{\hphantom{\neg A_1, \quad \neg A_2,}}^{B_1}} & & & \multicolumn{2}{c}{\overbrace{\hphantom{\neg A_5, \quad \neg A_6,}}^{B_3}} & & & \\
\{ & \neg A_1, & & \neg A_3, & \neg A_4, & \neg A_5, & \neg A_6, & \neg A_7 & \} & //\{\neg B_1, \neg B_3\} \\
\{ & \neg A_1, & & \neg A_3, & \neg A_4, & A_5, & & \neg A_7 & \} & //\{\neg B_1, B_3\} \\
\{ & \neg A_1, & & \neg A_3, & \neg A_4, & \neg A_5, & A_6, & \neg A_7 & \} & //\{\neg B_1, B_3\} \\
\{ & A_1, & \neg A_2, & \neg A_3, & \neg A_4, & \neg A_5, & \neg A_6, & \neg A_7 & \} & //\{\neg B_1, \neg B_3\} \\
\{ & A_1, & \neg A_2, & \neg A_3, & \neg A_4, & A_5, & & \neg A_7 & \} & //\{\neg B_1, B_3\} \\
\{ & A_1, & \neg A_2, & \neg A_3, & \neg A_4, & \neg A_5, & A_6, & \neg A_7 & \} & //\{\neg B_1, B_3\} \\
\{ & A_1, & A_2, & \neg A_3, & \neg A_4, & \neg A_5, & \neg A_6, & \neg A_7 & \} & //\{B_1, \neg B_3\} \\
\{ & A_1, & A_2, & \neg A_3, & \neg A_4, & A_5, & & \neg A_7 & \} & //\{B_1, B_3\} \\
\{ & A_1, & A_2, & \neg A_3, & \neg A_4, & \neg A_5, & A_6, & \neg A_7 & \} & //\{B_1, B_3\}
\end{array} \tag{6}$$

where $\mu^{\mathbf{A}\prime}$ (5) is the first in the list. $\lrcorner$

The example above shows an intrinsic problem of $\mathsf{CNF}_{\mathsf{Ts}}$ when used for enumeration: *if a minimal partial assignment $\mu^{\mathbf{A}}$ suffices to satisfy φ, this does not imply that $\mu^{\mathbf{A}}$ suffices to satisfy $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$*, i.e., that some $\eta^{\mathbf{B}}$ exists such that $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}}$satisfies $\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$ [18].

In fact, consider a generic non-CNF formula $\varphi(\mathbf{A})$ and a minimal partial truth assignment $\mu^{\mathbf{A}}$ that satisfies φ, and let φ_i be some sub-formula of φ which is not assigned a truth value by $\mu^{\mathbf{A}}$ – for instance, because φ_i occurs into some positive subformula $\varphi_i \vee \varphi_j$ and $\mu^{\mathbf{A}}$ satisfies φ_j. (In Example 1, $\mu^{\mathbf{A}} \overset{\text{def}}{=} \{\neg A_3, \neg A_4, \neg A_7\}$, $\varphi_i \overset{\text{def}}{=} (A_1 \wedge A_2)$ or $\varphi_i \overset{\text{def}}{=} (A_5 \vee A_6)$ respectively.) Then $\mathsf{CNF}_{\mathsf{Ts}}$ conjoins to the main formula the definition $(B_i \leftrightarrow \varphi_i)$, so that every satisfying partial truth assignment $\mu^{\mathbf{A}\prime}$ is forced to assign a truth value to φ_i and thus to some of its atoms, which may not occur in $\mu^{\mathbf{A}}$, so that $\mu^{\mathbf{A}\prime} \supset \mu^{\mathbf{A}}$. (In the example, the clauses in (3a) and (3c) force $\mu^{\mathbf{A}\prime}$ to assign a truth value also to $(A_1 \wedge A_2)$ and $(A_5 \vee A_6)$ respectively.)

Thus, by using $\mathsf{CNF}_{\mathsf{Ts}}$, instead of enumerating one minimal partial model $\mu^{\mathbf{A}}$ for φ, the solver may be forced to enumerate many partial models $\mu^{\mathbf{A}\prime}$ that are minimal for $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{Ts}}(\varphi)$ but not for φ, so that their number can be up to exponentially larger in the number of unassigned atoms in $\mu^{\mathbf{A}}$. In fact, each such model $\mu^{\mathbf{A}\prime}$ conjoins to $\mu^{\mathbf{A}}$ one of the (up to $2^{|\mathbf{A}|-|\mu^{\mathbf{A}}|}$) partial assignments which are needed to evaluate to either $\top$ or $\bot$ all unassigned

φ_i's. (E.g., in (6), the solver enumerates nine $\mu^{\mathbf{A}\prime}$s by conjoining $\mu^{\mathbf{A}}$ (2) with an exhaustive enumeration of partial assignments to A_1, A_2, A_5, A_6 that evaluate $(A_1 \wedge A_2)$ and $(A_5 \vee A_6)$ to either $\top$ or $\bot$.) This may drastically affect the effectiveness of the enumeration.

3.2 The impact of Plaisted and Greenbaum CNF transformation

We point out how the $\mathsf{CNF_{PG}}$ [17] can be used to solve these issues, but only in part. We first illustrate it with an example.

▶ **Example 2.** Consider the formula φ (1) and the minimal satisfying assignment $\mu^{\mathbf{A}}$ (2) as in Example 1. Suppose that φ is converted into CNF using $\mathsf{CNF_{PG}}$. Then, the following CNF formula is obtained:

$$\begin{array}{llll}
\mathsf{CNF_{PG}}(\varphi) \overset{\text{def}}{=} & & & \\
\quad (\neg B_1 \vee A_1) \wedge (\neg B_1 \vee A_2) & \wedge & //(B_1 \rightarrow (A_1 \wedge A_2)) & \text{(7a)} \\
\quad (B_2 \vee \neg A_3) \wedge (B_2 \vee \neg A_4) \wedge (\neg B_2 \vee A_3 \vee A_4) & \wedge & //(B_2 \leftrightarrow (A_3 \vee A_4)) & \text{(7b)} \\
\quad (B_3 \vee \neg A_5) \wedge (B_3 \vee \neg A_6) \wedge (\neg B_3 \vee A_5 \vee A_6) & \wedge & //(B_3 \leftrightarrow (A_5 \vee A_6)) & \text{(7c)} \\
\quad (\neg B_4 \vee B_2) \wedge (\neg B_4 \vee B_3) \wedge (B_4 \vee \neg B_2 \vee \neg B_3) & \wedge & //(B_4 \leftrightarrow (B_2 \wedge B_3)) & \text{(7d)} \\
\quad (\neg B_5 \vee B_4 \vee \neg A_7) \wedge (\neg B_5 \vee \neg B_4 \vee A_7) & \wedge & //(B_5 \rightarrow (B_4 \leftrightarrow A_7)) & \text{(7e)} \\
\quad (B_1 \vee B_5) & \wedge & & \text{(7f)}
\end{array}$$

We highlight that (7a) and (7e) are shorter than (3a) and (3e) respectively, since the corresponding sub-formulas occur only with positive polarity. Suppose, as in Example 1, that the solver finds the total truth assignment $\eta \overset{\text{def}}{=} \eta^{\mathbf{B}} \cup \eta^{\mathbf{A}}$ in (4). In this case, one possible output of the minimization procedure is the minimal partial truth assignment:

$$\mu^{\mathbf{A}\prime\prime} \overset{\text{def}}{=} \{\neg A_3, \neg A_4, \neg A_5, \neg A_6, \neg A_7\}. \tag{8}$$

This assignment is a strict sub-assignment of $\mu^{\mathbf{A}\prime}$ in (5), since the atom A_1 is not assigned. This is possible because the sub-formula $(A_1 \wedge A_2)$ is labelled by B_1 using a single implication, and the clauses representing $(B_1 \rightarrow (A_1 \wedge A_2))$ are satisfied by $\eta^{\mathbf{B}}(B_1) = \bot$ even without assigning A_1 and A_2. Nevertheless, the assignment $\mu^{\mathbf{A}}$ in (2) that satisfies φ *still does not satisfy* $\exists \mathbf{B}.\mathsf{CNF_{PG}}(\varphi)$.

Indeed, sub-formulas occurring with double polarity are labelled using double implications as for $\mathsf{CNF_{Ts}}$, raising the same problems as the latter. For instance, the sub-formula $(A_5 \vee A_6)$ occurs with double polarity, since it is under the scope of an "$\leftrightarrow$". Hence, the clauses in (7c) must be satisfied by assigning a truth value also to A_5 or A_6, and so the partial truth assignment $\mu^{\mathbf{A}}$ in (2) does not suffice to satisfy $\exists \mathbf{B}.\mathsf{CNF_{PG}}(\varphi)$. ⌟

The example above shows that $\mathsf{CNF_{PG}}$ has some advantage over $\mathsf{CNF_{Ts}}$ when enumerating partial assignments, but it overcomes its effectiveness issues only in part, *because a minimal assignment* $\mu^{\mathbf{A}}$ *satisfying* φ *may not suffice to satisfy* $\exists \mathbf{B}.\mathsf{CNF_{PG}}(\varphi)$, as with $\mathsf{CNF_{Ts}}$.

Consider, as in §3.1, a generic non-CNF formula $\varphi(\mathbf{A})$ and a partial truth assignment $\mu^{\mathbf{A}}$ that satisfies φ without assigning a truth value to some sub-formula φ_i. Suppose that φ_i occurs only positively in φ – for the negative case the reasoning is dual. (In Example 2, $\mu^{\mathbf{A}} \overset{\text{def}}{=} \{\neg A_3, \neg A_4, \neg A_7\}$, $\varphi_i \overset{\text{def}}{=} (A_1 \wedge A_2)$.) Since $\mathsf{CNF_{PG}}$ introduces only the clauses representing $(B_i \rightarrow \varphi_i)$ – and not those representing $(B_i \leftarrow \varphi_i)$ – the solver is no longer forced to assign a truth value to φ_i, because it suffices to assign $\eta^{\mathbf{B}}(B_i) = \bot$. (In the example, $(A_1 \wedge A_2)$ is labelled with B_1 in (7a).) In this case, φ_i plays the role of a "don't care" term, and this property allows for the enumeration of shorter partial assignments.

Nevertheless, a sub-formula can be "don't care" only if it occurs with single polarity. In fact, if φ_i occurs with double polarity – as it is the case, e.g., of sub-formulas under the scope of an "$\leftrightarrow$" – then φ_i is labelled with a double implication $(B_i \leftrightarrow \varphi_i)$, yielding the same drawbacks as with $\mathsf{CNF}_{\mathsf{Ts}}$. (In the example, $(A_5 \vee A_6)$ occurs with double polarity, and $\mu^{\mathbf{A}'}$ is forced to assign a truth value also to A_5 or A_6 to satisfy the clauses in (7c).)

Notice that, to maximize the benefits of $\mathsf{CNF}_{\mathsf{PG}}$, the sub-formulas that should be treated as "don't care" must have their label assigned to false. In practice, this can be achieved in part by instructing the solver to split on negative values in decision branches[2]. Even though the solver is not guaranteed to always assign to false the labels of "don't care" sub-formulas, we empirically verified that this heuristic provides a good approximation of this behaviour.

4 Enhancing enumeration via NNF preprocessing

In this section, we propose a possible solution to address the shortcomings of $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ CNF-izations in model enumeration, described in §3. We show that a simple preprocessing can avoid this situation. We transform first the input formula into an NNF DAG. In fact, NNF guarantees that each sub-formula occurs only positively, as every sub-formula φ_i occurring with double polarity is converted into two syntactically-different sub-formulas $\varphi_i^+ \overset{\text{def}}{=} \mathsf{NNF}(\varphi_i)$ and $\varphi_i^- \overset{\text{def}}{=} \mathsf{NNF}(\neg\varphi_i)$ – each occurring only positively – which are then labelled – with single implications – with two distinct atoms B_i^+ and B_i^- respectively. To improve the efficiency of the enumeration procedure without affecting its outcome, we also add the clauses $(\neg B_i^+ \vee \neg B_i^-)$ when both B_i^+ and B_i^- are introduced, which prevent the solver from assigning both B_i^+ and B_i^- to true, and thus from exploring inconsistent search branches.

We remark that even with this preprocessing we produce a linear-size CNF encoding, since the $\mathsf{NNF}(\varphi)$ DAG has linear size w.r.t. φ (see §2.1), and $\mathsf{CNF}_{\mathsf{PG}}$ introduces one label definition for each DAG node, each consisting of 1 or 2 clauses. We illustrate the benefit of this additional preprocessing with the following example.

▶ **Example 3.** Consider the formula φ of Example 1. By converting it into NNF, we obtain:

$$\varphi' \overset{\text{def}}{=} \overbrace{(A_1 \wedge A_2)}^{B_1} \vee \underbrace{(\overbrace{(\underbrace{(\overbrace{(\neg A_3 \wedge \neg A_4)}^{B_2^-} \vee \overbrace{(\neg A_5 \wedge \neg A_6)}^{B_3^-})}_{B_4^-} \vee A_7)}^{B_5} \wedge \overbrace{(\underbrace{(\overbrace{(A_3 \vee A_4)}^{B_2^+} \wedge \overbrace{(A_5 \vee A_6)}^{B_3^+})}_{B_4^+} \vee \neg A_7)}^{B_6})}_{B_7} \tag{9}$$

[2] To exploit this heuristic also for sub-formulas occurring only negatively, the latter can be labelled with a negative label $\neg B_i$ as $(\neg B_i \leftarrow \varphi_i)$.

Suppose, then, that the formula is converted into CNF using $\mathsf{CNF}_{\mathsf{PG}}$. Then, the following CNF formula is obtained:

$$\begin{aligned}
\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi)) \overset{\text{def}}{=} & (\neg B_1 \vee A_1) \wedge (\neg B_1 \vee A_2) \wedge && //(B_1 \rightarrow (A_1 \wedge A_2)) && \text{(10a)}\\
& (\neg B_2^- \vee \neg A_3) \wedge (\neg B_2^- \vee \neg A_4) \wedge && //(B_2^- \rightarrow (\neg A_3 \wedge \neg A_4)) && \text{(10b)}\\
& (\neg B_3^- \vee \neg A_5) \wedge (\neg B_3^- \vee \neg A_6) \wedge && //(B_3^- \rightarrow (\neg A_5 \wedge \neg A_6)) && \text{(10c)}\\
& (\neg B_4^- \vee B_2^- \vee B_3^-) \wedge && //(B_4^- \rightarrow (B_2^- \vee B_3^-)) && \text{(10d)}\\
& (\neg B_5 \vee B_4^- \vee A_7) \wedge && //(B_5 \rightarrow (B_4^- \vee A_7)) && \text{(10e)}\\
& (\neg B_2^+ \vee A_3 \vee A_4) \wedge && //(B_2^+ \rightarrow (A_3 \vee A_4)) && \text{(10f)}\\
& (\neg B_3^+ \vee A_5 \vee A_6) \wedge && //(B_3^+ \rightarrow (A_5 \vee A_6)) && \text{(10g)}\\
& (\neg B_4^+ \vee B_2^+) \wedge (\neg B_4^+ \vee B_3^+) \wedge && //(B_4^+ \rightarrow (B_2^+ \wedge B_3^+)) && \text{(10h)}\\
& (\neg B_6 \vee B_4^+ \vee \neg A_7) \wedge && //(B_6 \rightarrow (B_4^+ \vee \neg A_7)) && \text{(10i)}\\
& (\neg B_7 \vee B_5) \wedge (\neg B_7 \vee B_6) \wedge && //(B_7 \rightarrow (B_5 \wedge B_6)) && \text{(10j)}\\
& (B_1 \vee B_7) \wedge && && \text{(10k)}\\
& (\neg B_2^+ \vee \neg B_2^-) \wedge && && \text{(10l)}\\
& (\neg B_3^+ \vee \neg B_3^-) \wedge && && \text{(10m)}\\
& (\neg B_4^+ \vee \neg B_4^-) && && \text{(10n)}
\end{aligned}$$

Suppose, e.g., that the solver picks non-deterministic choices, deciding the atoms in the order $\{B_1, A_1, A_2, B_3^-, A_5, A_6, B_2^-, A_3, A_4, B_4^-, B_5, A_7, B_2^+, B_3^+, B_4^+, B_6, B_7\}$, branching with a negative value first. Then, the first total truth assignment found is:

$$\eta \overset{\text{def}}{=} \{\underbrace{\neg B_1, B_2^-, \neg B_3^-, B_4^-, B_5, \neg B_2^+, \neg B_3^+, \neg B_4^+, B_6, B_7}_{\eta^{\mathbf{B}}}, \underbrace{\neg A_1, \neg A_2, \neg A_3, \neg A_4, \neg A_5, \neg A_6, \neg A_7}_{\eta^{\mathbf{A}}}\}. \tag{11}$$

In this case, the minimization procedure returns $\mu^{\mathbf{A}} \overset{\text{def}}{=} \{\neg A_3, \neg A_4, \neg A_7\}$ as in (5), achieving full minimization. With this additional preprocessing, in fact, the solver is no longer forced to assign a truth value to A_5 or A_6. This is possible because, even though $(A_5 \vee A_6)$ occurs with double polarity in φ, in $\mathsf{NNF}(\varphi)$ its positive and negative occurrences are converted into $(A_5 \vee A_6)$ and $(\neg A_5 \wedge \neg A_6)$ respectively. Since they appear as two syntactically-different sub-formulas, $\mathsf{CNF}_{\mathsf{PG}}$ labels them – with single implications – using two different atoms B_3^+ and B_3^- respectively. This allows the solver to find a model η that assigns both B_3^- and B_3^+ to false. Hence, the clauses in (10c) and (10g) are satisfied even without assigning A_5 and A_6, and thus these atoms can be dropped by the minimization procedure. ⌟

The key idea behind this additional preprocessing is that each sub-formula of $\mathsf{NNF}(\varphi)$ occurs only positively, so that $\mathsf{CNF}_{\mathsf{PG}}$ labels them with single implications, and the solver is no longer forced to assign them a truth value. Consider a sub-formula φ_i that occurs with double polarity in φ. In $\mathsf{NNF}(\varphi)$ the two subformulas $\varphi_i^+ \overset{\text{def}}{=} \mathsf{NNF}(\varphi_i)$ and $\varphi_i^- \overset{\text{def}}{=} \mathsf{NNF}(\neg\varphi_i)$ occur only positively. Then, instead of adding $(B_i \leftrightarrow \varphi_i)$, we add $(B_i^+ \rightarrow \varphi_i^+) \wedge (B_i^- \rightarrow \varphi_i^-)$, and the solver can find a truth assignment η that assigns both B_i^- and B_i^+ to false. (In Example 3, instead of $(B_3 \leftrightarrow (A_5 \vee A_6))$ we add $(B_3^+ \rightarrow (A_5 \vee A_6))$ and $(B_3^- \rightarrow (\neg A_5 \wedge \neg A_6))$.) Thus, the clauses deriving from φ_i can be satisfied even without assigning a truth value to φ_i, whose atoms can be dropped by the minimization procedure – provided that they are not forced to be assigned by some other sub-formula of φ. (In the example, by setting $\eta^{\mathbf{B}}(B_3^+) = \eta^{\mathbf{B}}(B_3^-) = \bot$, the clauses in (10c) and (10g) are satisfied even without assigning A_5 and A_6.)

We have the following general fact: *every partial model* $\mu^{\mathbf{A}}$ *for* φ *is also a model for* $\exists \mathbf{B}.\mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$, that is, if $\mu^{\mathbf{A}} \models \varphi$, then there exists $\eta^{\mathbf{B}}$ s.t. $\mu^{\mathbf{A}} \cup \eta^{\mathbf{B}} \models \mathsf{CNF}_{\mathsf{PG}}(\mathsf{NNF}(\varphi))$. (The vice versa holds trivially.) A complete formal proof of this fact is presented in an extended version of this paper [13]. Intuitively, it is easy to see that the suitable $\eta^{\mathbf{B}}$ is defined so that, for each sub-formula φ_i of φ, if φ_i is made true, false or is unassigned by $\mu^{\mathbf{A}}$, then $\langle \eta^{\mathbf{B}}(B_i^+), \eta^{\mathbf{B}}(B_i^-) \rangle$ is $\langle \top, \bot \rangle$, $\langle \bot, \top \rangle$, or $\langle \bot, \bot \rangle$ respectively.

We stress the fact that this does not guarantee that the enumeration procedure always finds this $\eta^{\mathbf{B}}$, but only that such $\eta^{\mathbf{B}}$ exists. Ad-hoc enumeration heuristics should be investigated.

▶ Remark 4. We notice that the pre-conversion into NNF is typically never used in plain SAT *solving*, because it causes the unnecessary duplication of labels B_i^+ and B_i^-, with extra overhead and no benefit for the solver.

5 Experimental evaluation

In this section, we experimentally evaluate the impact of different CNF-izations on the AllSAT task. In order to compare them on a fair ground, we have implemented a base version of each from scratch in PySMT [7], avoiding specific optimizations done by the solvers. We used MathSAT [6] as a SAT enumerator, because it implements the enumeration strategy by [12] described in §2.2. We set the options -dpll.branching_initial_phase=0 to split on the false branch first and -dpll.branching_cache_phase=2 to enable phase caching.

Experiments run on an Intel Xeon Gold 6238R @ 2.20GHz 28 Core machine with 128 GB of RAM and running Ubuntu Linux 20.04. For each instance, we set a timeout of 1200s.

5.1 Datasets description

We consider three sets of benchmarks of non-CNF formulas coming from different sources, both synthetic and real-world. In the first set of benchmarks, we generate random Boolean formulas by nesting Boolean operators up to a fixed depth. The second dataset consists of Boolean formulas encoding properties of ISCAS'85 circuits [4, 8, 20]. As a third set of problems, we consider formulas encoding Booleanized Weighted Model Integration (WMI) problems [14, 15, 19].

The synthetic benchmarks

The synthetic benchmarks are generated by nesting Boolean operators $\wedge, \vee, \leftrightarrow$ until some fixed depth d. Internal and leaf nodes are negated with 50% probability. Operators in internal nodes are chosen randomly, giving less probability to the $\leftrightarrow$ operator. In particular, $\leftrightarrow$ is chosen with a probability of 10%, whereas the other two are chosen with an equal probability of 45%. We generated 100 synthetic instances over a set of 20 Boolean atoms and depth $d = 8$.

The circuits benchmarks

The ISCAS'85 benchmarks are a set of 10 combinatorial circuits used in test generation, timing analysis, and technology mapping [4]. They have well-defined, high-level structures and functions based on multiplexers, ALUs, decoders, and other common building blocks [8]. We generated random instances as described in [20]. In particular, for each circuit, we constrained 60%, 70%, 80%, 90% and 100% of the outputs to either 0 or 1, for a total of 250 instances.

The WMI benchmarks

WMI problems are generated using the procedure described in [19]. Specifically, the paper addresses the problem of enumerating all the different paths of the weight function by encoding it into a skeleton formula. Each instance consists of a skeleton formula of a randomly-generated weight function, where the conditions are only over Boolean atoms. Since the conditions are non-atomic, the resulting formula is not in CNF, and thus we preprocess it with the different CNF-izations before enumerating its models. We generate 10 instances for each depth value 3, 5, 7, 9, each instance involving 10 Boolean atoms and no real variable, for a total of 40 problems.

We remark on two aspects of these benchmarks. First, we have chosen to have Boolean-only weight conditions in order to better analyse the capacity of Boolean reasoning of the solver with the different transformations, without additional factors brought by the SMT component. Nevertheless, we expect to have similar outcomes also for formulas involving both Boolean and SMT($\mathcal{LRA}$) atoms. Notice that these can still be meaningful WMI instances, as the $\mathcal{LRA}$ component may be constrained by the rest of the formula. Second, these formulas contain existentially quantified SMT($\mathcal{EUF}$) atoms, so that we enumerate $\exists \mathbf{y}.\varphi(\mathbf{A}, \mathbf{y})$ by projecting the models of φ over the relevant atoms $\mathbf{A}$ [19].

5.2 Results

Figures 1, 2, and 3 show the results of the experiments on the synthetic, ISCAS'85 and WMI benchmarks, respectively. For each group of benchmarks, we report a set of scatter plots to compare $\mathsf{CNF}_{\mathsf{Ts}}$, $\mathsf{CNF}_{\mathsf{PG}}$ and $\mathsf{NNF} + \mathsf{CNF}_{\mathsf{PG}}$ in terms of number models, in the first row, and execution time, in the second row. Notice the logarithmic scale of the axes!

In §5.3 we also report the CDF of the execution time for plain SAT solving on the same group of benchmarks. We see from the results that, unlike with enumeration, the pre-conversion into NNF has no benefit for plain solving, as we observed in Remark 4.

The synthetic benchmarks

The results on the synthetic benchmarks are shown in Figure 1. All the problems were solved for all the encodings within the timeout. The plots show that $\mathsf{CNF}_{\mathsf{PG}}$ performs better than $\mathsf{CNF}_{\mathsf{Ts}}$, since it enumerates fewer models (first row) in less time (second row) on every instance. Furthermore, the combination of NNF and $\mathsf{CNF}_{\mathsf{PG}}$ yields by far the best results, drastically reducing the number of models and the execution time by orders of magnitude w.r.t. both $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$.

The circuits benchmarks

Figure 2 shows the performance of the different CNF-izations in the circuits benchmarks. The timeouts are represented by the points on the dashed lines. First, we notice that $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ have very similar behaviour, both in terms of execution time and number of models. The reason is that in circuits, it is typical to have a lot of sharing of sub-formulas. Since we constrain the outputs to be 0 or 1 at random [20], most of the sub-formulas occur with double polarity, so that the two encodings are very similar, if not identical. Second, we notice that by converting the formula into NNF before applying $\mathsf{CNF}_{\mathsf{PG}}$ the enumeration is much more effective, as a much smaller number of models is enumerated, with only a few outliers. The fact that for some instances $\mathsf{NNF} + \mathsf{CNF}_{\mathsf{PG}}$ takes a little more time can be caused by the fact that it can produce a formula that is up to twice as large and contains up

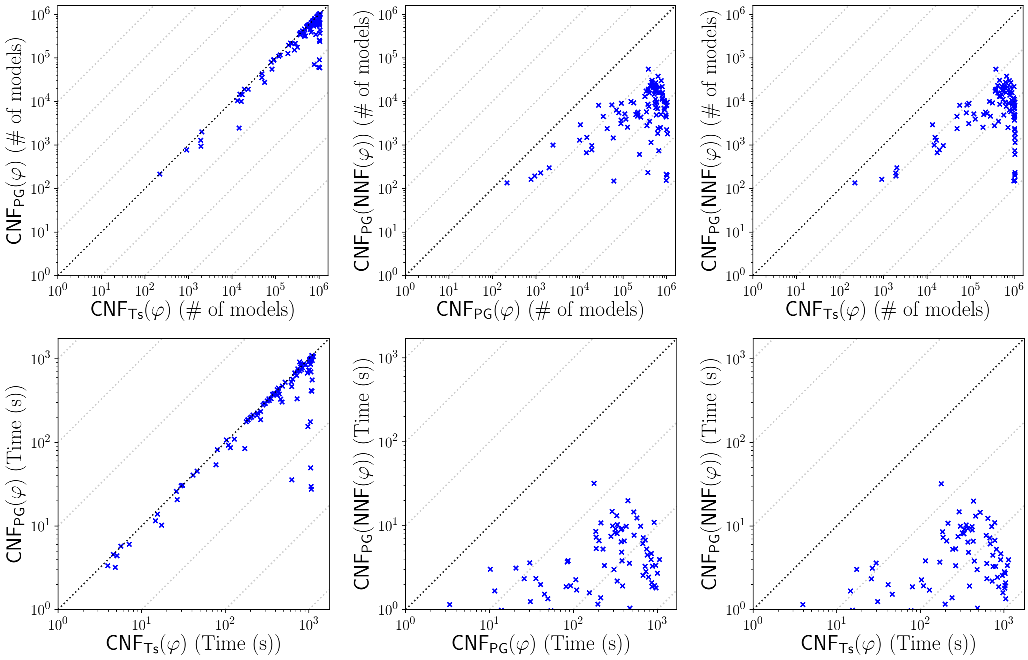


Figure 1 Set of scatter plots comparing the different CNF-izations on the synthetic benchmarks. The first and second rows compare them in terms of number of models and execution time, respectively. All the axes are on a logarithmic scale. In these problems, there were no timeouts.

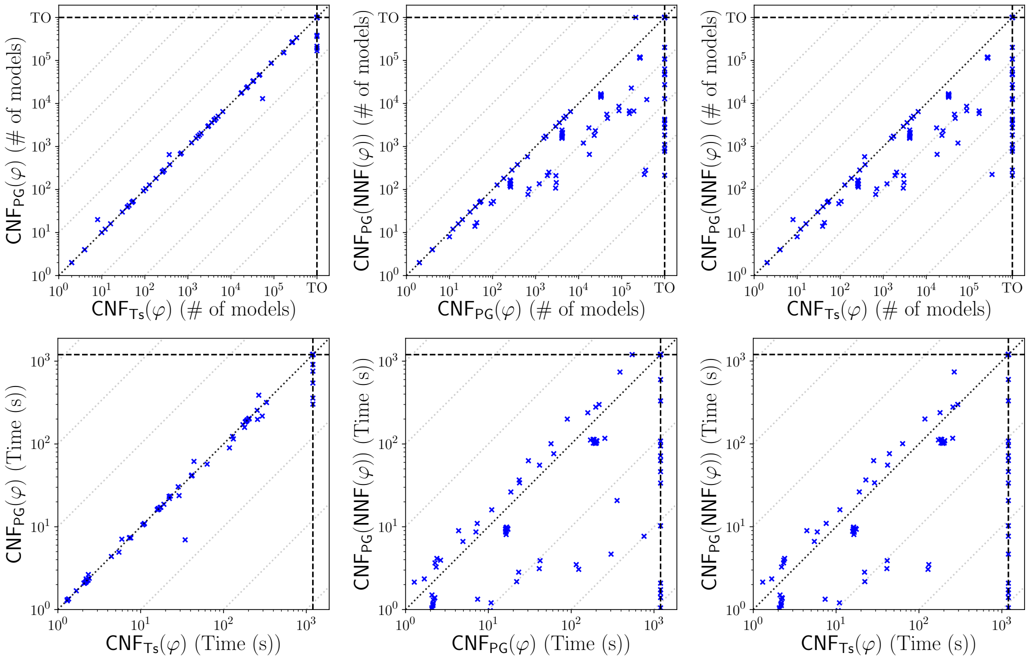


Figure 2 Set of scatter plots comparing the different CNF-izations on the circuits benchmarks. The first and second rows compare them in terms of number of models and execution time, respectively. All the axes are on a logarithmic scale. In these problems, the CNF$_{Ts}$, CNF$_{PG}$ and NNF + CNF$_{PG}$ reported 49, 44 and 27 timeouts, respectively, represented by the points on the dashed lines.

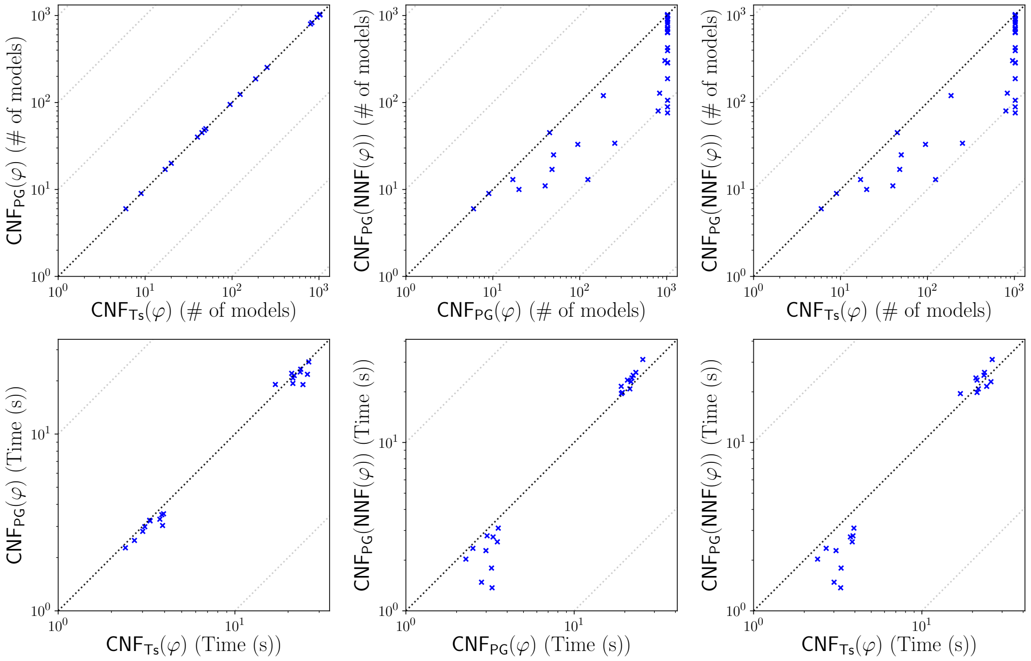


Figure 3 Set of scatter plots comparing the different CNF-izations on the WMI benchmarks. The first and second rows compare them in terms of number of models and execution time, respectively. All the axes are on a logarithmic scale. In these problems, there were no timeouts.

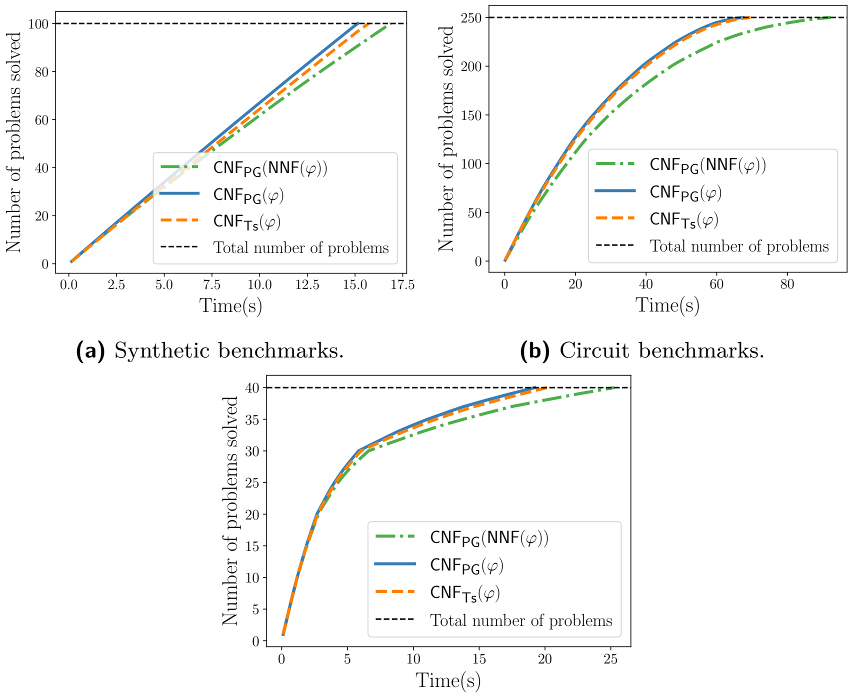
to twice as many label atoms as the other two encodings, increasing the time to find the assignments. Notice also that, even enumerating a smaller number of models at a price of a small time-overhead can be beneficial in many applications, for instance in WMI [14, 15, 19].

The WMI benchmarks

The plots in Figure 3 compare the different CNF-izations in the WMI benchmarks in terms of number of models and time. All the problems were solved for all the encodings within the timeout. In these benchmarks, most of the sub-formulas occur with double polarity, so that $\mathsf{CNF}_{\mathsf{Ts}}$ and $\mathsf{CNF}_{\mathsf{PG}}$ encodings are almost identical, and they obtain very similar results in both metrics. The advantage is significant, instead, if the formula is converted into NNF upfront, since by using $\mathsf{NNF} + \mathsf{CNF}_{\mathsf{PG}}$ the solver enumerates a smaller number of models. In this application, it is crucial to enumerate as few models as possible, since for each model an integral must be computed, which is a very expensive operation [14, 15, 19].

5.3 Comparing the CNF encodings for SAT solving

In order to confirm the statement in Remark 4, in the CDFs in Figure 4 we compare the different CNF encodings for *plain SAT solving* on the same benchmarks. Even though these problems are very small for plain solving and SAT solvers deal with them very efficiently, we can see that converting the formula into NNF before applying $\mathsf{CNF}_{\mathsf{PG}}$ brings no advantage, and solving is uniformly slower than with $\mathsf{CNF}_{\mathsf{PG}}$ or $\mathsf{CNF}_{\mathsf{Ts}}$. This shows that our novel technique works specifically for enumeration but not for solving, as expected.



(a) Synthetic benchmarks.

(b) Circuit benchmarks.

(c) WMI benchmarks.

Figure 4 CDF of the time taken for plain SAT solving using the different CNF transformations. The y-axis reports the instances for which the enumeration finished within the cumulative time on the x-axis.

6 Conclusions and future work

We have presented a theoretical and empirical analysis of the impact of different CNF-ization approaches on SAT enumeration. We have shown how the most popular transformations conceived for SAT solving, namely the Tseitin and the Plaisted and Greenbaum CNF-izations, prevent the solver from producing short partial assignments, thus seriously affecting the effectiveness of the enumeration. To overcome this limitation, we have proposed to preprocess the formula by converting it into NNF before applying the Plaisted and Greenbaum transformation. We have shown, both theoretically and empirically, that the latter approach can fully overcome the problem and can drastically reduce both the number of partial assignments and the execution time.

As future research directions, we plan to further investigate the impact of CNF conversion also on disjoint SMT enumeration. We expect that in this domain the impact can be even more relevant, since in SMT multiple instances of the same theory atoms are typically rarer than for atoms in the Boolean case. Also, disjoint SMT enumeration has a fundamental role in Weighted Model Integration [14, 15, 19], an important framework for probabilistic inference in hybrid domains. Hence, we believe that our contribution can have a great impact on this application, where non-CNF formulas occur frequently. Finally, we think that work should be done to understand the impact on enumeration with repetitions, i.e. where models may not be disjoint, for instance in Predicate Abstraction [12].

References

1 A. Biere and H. van Maaren. *Handbook of Satisfiability: Second Edition.* IOS Press, May 2021.

2 Magnus Björk. Successful SAT Encoding Techniques. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4):189–201, July 2009.

3 Thierry Boy de la Tour. An Optimality Result for Clause Form Translation. *Journal of Symbolic Computation*, 14(4):283–301, October 1992.

4 F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Proceedings of IEEE International Symposium Circuits and Systems (ISCAS 85)*, pages 677–692. IEEE Press, Piscataway, N.J., 1985.

5 Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. Approximate Counting in SMT and Value Estimation for Probabilistic Programs. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems – Volume 9035*, pages 320–334, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

6 Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 93–107. Springer, 2013.

7 Marco Gario and Andrea Micheli. PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In *SMT Workshop 2015*, 2015.

8 M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering. *IEEE Design & Test of Computers*, 16(3):72–80, 1999.

9 Markus Iser, Carsten Sinz, and Mana Taghdiri. Minimizing Models for Tseitin-Encoded SAT Instances. In *Theory and Applications of Satisfiability Testing – SAT 2013*, volume 7962, pages 224–232. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. Series Title: LNCS.

10 Paul Jackson and Daniel Sheridan. The Optimality of a Fast CNF Conversion and its Use with SAT. In *Theory and Applications of Satisfiability Testing – SAT 2004*, 2004.

11 Elias Kuiter, Sebastian Krieter, Chico Sundermann, Thomas Thüm, and Gunter Saake. Tseitin or not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In *37th IEEE/ACM Int. Conference on Automated Software Engineering*, pages 1–13. ACM, 2022.

12 Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. SMT Techniques for Fast Predicate Abstraction. In *Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 424–437. Springer Berlin Heidelberg, 2006.

13 Gabriele Masina, Giuseppe Spallitta, and Roberto Sebastiani. On CNF Conversion for SAT Enumeration, June 2023. `arXiv:2303.14971`.

14 Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. Efficient Weighted Model Integration via SMT-Based Predicate Abstraction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 720–728, 2017.

15 Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. Advanced SMT techniques for Weighted Model Integration. *Artificial Intelligence*, 275(C):1–27, 2019.

16 Sibylle Möhle, Roberto Sebastiani, and Armin Biere. Four Flavors of Entailment. In *Theory and Applications of Satisfiability Testing – SAT 2020*, Lecture Notes in Computer Science, pages 62–71. Springer, 2020.

17 David A. Plaisted and Steven Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

18 Roberto Sebastiani. Are You Satisfied by This Partial Assignment?, 2020. `arXiv:2003.04225`.

19 Giuseppe Spallitta, Gabriele Masina, Paolo Morettin, Andrea Passerini, and Roberto Sebastiani. SMT-based Weighted Model Integration with Structure Awareness. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence*, volume 180, pages 1876–1885, 2022.

20 Abraham Temesgen Tibebu and Goerschwin Fey. Augmenting All Solution SAT Solving for Circuits with Structural Information. In *IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 117–122, 2018.

21 G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-70*, pages 466–483. Springer, 1983.

Bounds on BDD-Based Bucket Elimination

Stefan Mengel
Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France

Abstract

We study BDD-based bucket elimination, an approach to satisfiability testing using variable elimination which has seen several practical implementations in the past. We prove that it allows solving the standard pigeonhole principle formulas efficiently, when allowing different orders for variable elimination and BDD-representations, a variant of bucket elimination that was recently introduced. Furthermore, we show that this upper bound is somewhat brittle as for formulas which we get from the pigeonhole principle by restriction, i.e., fixing some of the variables, the same approach with the same variable orders has exponential runtime. We also show that the more common implementation of bucket elimination using the same order for variable elimination and the BDDs has exponential runtime for the pigeonhole principle when using either of the two orders from our upper bound, which suggests that the combination of both is the key to efficiency in the setting.

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Constraint and logic programming

Keywords and phrases Bucket Elimination, Binary Decision Diagrams, Satisfiability, Complexity

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.16

Funding This work has been partly supported by the PING/ACK project of the French National Agency for Research (ANR-18-CE40-0011).

1 Introduction

We analyze several aspects of a simple approach to propositional satisfiability called *bucket elimination* based on binary decision diagrams (BDDs) [5]. It was originally introduced by Pan and Vardi [15], and works, given a CNF F, as follows: first translate all clauses of F into BDDs, all having the same variable order. Then, along another variable order, conjoin all BDDs that contain the current variable x, eliminate x in the result of the conjoin operation by existential quantification, add the resulting BDD to the current set of BDDs, and finally delete all BDDs containing x. The end result is a BDD representing one of the constants 1 or 0, depending on if F is satisfiable or not. The algorithm is often described by putting the BDDs in *buckets* treated in the variable order as in the pseudocode Algorithm 1. It is not hard to see that this approach decides satisfiability of all CNF-formulas correctly. We remark in passing that bucket elimination has also been used as a general approach for reasoning in artificial intelligence [12]. In particular, in the context of propositional satisfiability one can implement ordered resolution, also called Davis-Putnam resolution [11], with it, which leads to an algorithm that is similar to what we described above [17] but uses CNF-formulas to represent intermediate results and not BDDs. In the remainder, we will only focus on bucket elimination that is based on BDDs.

Several SAT-solvers using bucket elimination have been implemented [15, 14, 6], also motivated by a relation to extended resolution which allows extracting clausal refutations of CNF-formulas efficiently from runs of bucket elimination.

In this paper, we aim to get a theoretical understanding of the strength of bucket elimination. We first prove that the approach is powerful enough to efficiently solve the well-known pigeonhole principle formulas PHP_n which are hard for other techniques, in particular resolution [13]. Our bound confirms recent experimental results for a different

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 16; pp. 16:1–16:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Algorithm 1 BDD-based bucket elimination for CNF-formulas.

```
Input: clauses $C_1, \dots, C_m$ in variable set $X$, elimination variable order $\pi$
for $x \in X$ do
    create empty bucket $B_x$
for $i = 1, \dots, m$ do
    compute a BDD for $C_i$, put it into $B_y$ where $y$ is first variable in order $\pi$ in $C_i$
for $x \in X$ in order $\pi$ do
    compute BDD $D$ by iteratively conjoining all BDDs in $B_x$
    if $D$ is constant 0-BDD then
        return 0
    compute a BDD $D'$ computing $\exists x D$ and put it into $B_y$ where $y$ is first variable in
      order $\pi$ in $D'$
return 1
```

encoding that was specifically chosen to make the algorithm efficient [9, 10]. We here show that also for the standard encoding, there is a choice of variable orders with which bucket elimination can efficiently solve pigeonhole principle formulas.

We then go on showing that the upper bound for PHP_n is in a sense brittle: one can restrict the formula PHP_n by assigning some of its variables, resulting in a formula on which bucket elimination with the same variable orders as before takes exponential time. This is surprising since fixing some of the variables reduces the search space and thus should make the problem easier. However, in the case of bucket elimination it has the opposite effect, making the runtime explode. This suggests that bucket elimination is not very stable under small variations of the input.

The final part of this paper is motivated by the fact that the pigeonhole principle has been used as a benchmark also in [15, 6] where bucket elimination was shown to be practically inefficient. The difference between our result and [10] on the one hand and [15, 6] on the other hand is that the latter, as also the implementation of [14], consider the same variable order for the variable elimination and the order in the BDDs. In contrast, in our result and the current public version of the implementation of [6, 9, 10][1] two different orders may be chosen. To explore the impact of this change, we consider bucket elimination for PHP_n where only one of the variable orders we use in our upper bound is used. We show that in both cases the variant that uses only one order has exponential runtime, which shows that to efficiently solve PHP_n the combination of the two orders is crucial and is more powerful than each of them individually.

Our results can also be seen in the context of BDD-based proof systems, more specifically, they are close to results on the proof system $\mathrm{OBDD}(\wedge, \exists)$ [2] which allows general conjunction and variable elimination without any scheduling restrictions. It was shown in [8] that there are polynomial size refutations of the pigeonhole principle in this system. This also follows from our result, which can be interpreted as working in a restricted fragment of $\mathrm{OBDD}(\wedge, \exists)$. We remark also that [7] claims that the proofs in [8] can be implemented in the algorithm of [15]. However, this seems to be not the case due to the order restrictions of that algorithm which are not respected in the proof. It is however possible that, after rearranging the operations, the proof in [8] could be implemented with two orders, similarly to our result.

[1] `https://github.com/rebryant/pgbdd`

2 Preliminaries

We use the usual integer interval notation, e.g. $[n] := \{1, \dots, n\}$ and $[m, n] := \{m, m + 1, \dots, n-1, n\}$. When speaking of graphs, we mean finite, simple, undirected graphs. We write $G = (A, B, E)$ for a bipartite graph with color classes A and B and edge set E.

We assume that the reader is familiar with the basics of propositional satisfiability, in particular CNF-formulas, see e.g. the introductory chapters of [4]. Given a CNF-formula F and a partial assignment a, we call the *restriction* of F by a the CNF which we get by fixing the variables according to a and simplifying, i.e., we delete all clauses that are satisfied by a and from the other clauses all literals that are falsified.

An *(ordered) binary decision diagram* (short *BDD* or OBDD) is a graph-based representation of Boolean functions as follows [5]: a BDD over a variable set X consists of a directed acyclic graph with one source and two sinks. The sinks are labeled 0 and 1, respectively, while all other nodes are labeled by variables from X. Every node but the sinks has two out-going edges, called 0-edge and 1-edge, respectively. Given an assignment a to X, we construct a source-sink path in the BDD starting in the source and iteratively following the $a(x)$-edge to the next node, where x is the label of the current node. Eventually, we end up in a sink whose label is the value computed by the BDD on a. This way, the BDD specifies a Boolean value for every assignment to X and thus defines a Boolean function. BDDs are required to be ordered as follows: there is an order π on X such that whenever there is an edge from a node labeled by x to a node labeled by y, then x appears before y in π. It follows that on every source-sink path one encounters every variable at most once.

It will sometimes be convenient to reason with complete BDDs which are BDDs in which all source-sink paths contain all variables as labels. The *width* of a complete BDD is defined as the maximal number of nodes that are labeled by the same variable. Clearly, a complete BDD in n variables and of width w has size at most $O(nw)$. Moreover, it is well known that when conjoining two BDDs with the same variable order and width w_1 and w_2, respectively, the result has the same order and width at most $w_1 \cdot w_2$.

We will use the following known lower bound, see e.g. [1, Section 6]; for the convenience of the reader, we give a self-contained proof in the appendix.

▶ **Lemma 1.** *Every BDD computing $\bigwedge_{i\in[n]} x_i \vee y_i$ with a variable order in which every x_i comes before every y_j has at least 2^n nodes.*

3 A Polynomial Upper Bound for the Pigeonhole Principle

We consider SAT-encodings of pigeonhole problems on bipartite graphs $G = (A, B, E)$[2]. We assume that $|B| > |A|$, so there is no perfect matching in the graph. In the graphs we consider, we will have $A = [n]$ and $B = [n + 1]$. We encode the non-existence of a perfect matching by generalizing the usual direct encoding of the pigeonhole principle: for every edge $ij \in E$, we introduce a variable $p_{i,j}$ which encodes if the edge ij is put into a matching or not. For every $i \in A$, we encode by an at-most-one constraint

$$\mathrm{AMO}_i := \bigwedge_{j,k\in N(i), j\neq k} \bar{p}_{i,j} \vee \bar{p}_{i,k},$$

[2] We remark that the same formulas are called *bipartite perfect matching benchmarks* in [9, 10], but since the name *perfect matching principle* is used for a related but different class of formulas in proof complexity [16], we follow the notation from [3] here and speak of *pigeonhole formulas* to avoid confusion.

that at most one vertex from B is matched to i. Here, $N(i)$ is the neighborhood of i in G, i.e., the set of vertices connected to i by an edge. For every $j \in B$, we add a clause

$$\mathrm{ALO}_j := \bigvee_{i \in N(j)} p_{i,j}$$

encoding the fact that j must be matched to one of its neighbors in A.

The pigeonhole formula for G is then

$$G\text{-PHP} := \bigwedge_{j \in B} \mathrm{ALO}_j \wedge \bigwedge_{i \in A} \mathrm{AMO}_i.$$

We recover the usual pigeonhole problem formula PHP_n by considering the complete bipartite graph $K_{n,n+1} = ([n], [n+1], [n] \times [n+1])$. Conversely, we get G-PHP from PHP_n by the restriction that sets the variables $p_{i,j}$ for $ij \notin E$ to 0.

It is useful to consider the variables $p_{i,j}$ of PHP_n organized in a matrix where, as usual, i gives the row index while j gives the column index. Note that with this convention, ALO_j only has variables in column j while AMO_i only has variables in row i.

We consider two orders on the variables in PHP_n: the *row-wise order*

$$\pi_r := p_{1,1}, p_{1,2}, \dots, p_{1,n+1}, p_{2,1}, \dots p_{n,n+1}$$

that we get by reading the variable matrix row by row and the *column-wise order*

$$\pi_c := p_{1,1}, p_{2,1}, \dots, p_{n,1}, p_{1,2}, \dots p_{n,n+1}$$

that we get by reading the variable matrix column by column. We consider the same orders for subgraphs G of $K_{n,n+1}$ by simply deleting the variables of edges not in G.

▶ **Theorem 1.** *Bucket elimination in which all BDDs have order π_r and the elimination proceeds in order π_c refutes PHP_n in polynomial time.*

Proof. We will polynomially bound the size of all BDDs constructed by the algorithm; since all BDD operations we use can be performed in time polynomial in the BDD size [5], the result then follows directly. In this we tacitly also use the fact that all operations on BDDs we use the constructed BDDs can be assumed to be a minimal size for the variable order due to canonicity of BDDs. We first analyze the BDDs that result from the respective quantification steps (Line 9 in Algorithm 1). We denote by $F'_{i,j}$ the function computed by the BDD in which we quantify $p_{i,j}$. By $F_{i,j}$ we denote the CNF formula that is the conjunction of all clauses that have been conjoined before this elimination step. Observe that we get $F'_{i,j}$ from $F_{i,j}$ by quantifying all variables up to $p_{i,j}$ in π_c. Moreover, if $p_{i,j}$ is before $p_{i',j'}$ in π_c, then the clauses in $F_{i,j}$ are a subset of those in $F_{i',j'}$.

$F_{i,j}$ consists of all clauses of PHP_n that have a variable up to $p_{i,j}$ in the order π_c, so

1. the clauses ALO_k for all $k \leq j$,
2. the clauses $\bar{p}_{i',k} \vee \bar{p}_{i',\ell}$, for $i' \in [n]$ and $1 \leq k < \ell \leq n+1$, $k < j$, and
3. the clauses $\bar{p}_{i',j} \vee \bar{p}_{i',k}$ for $i' \in [i]$ and $k \in [n+1], k \neq j$.

Remember that we get $F'_{i,j}$ from $F_{i,j}$ by quantifying the variables up to $p_{i,j}$ in the order π_c. We will show that $F'_{i,j}$ can be encoded by a small BDD. We first consider the case $j = 1$.

▷ Claim 2. An assignment a' satisfies $F'_{i,1}$ if and only if

- a' sets one of the $p_{i',1}$ with $i' \in [i+1, n]$ to 1, or
- there is an $i^* \in [i]$ such that all $p_{i^*,j}$ with $j \in [2, n+1]$ take the value 0 in a'.

Proof. Assume first that there is a $p_{i',1}$ with $i' \in [i+1, n]$ set to 1 by a'. We extend a' to an assignment a of $F_{i,1}$ by setting all quantified variables $p_{k,1}$ for $k \in [i]$ to 0. Then ALO_1 is satisfied by $p_{i',1}$ and the $p_{k,1}$ for $k \in [i]$ satisfy the clauses in 3. Since $F_{i,1}$ does not have clauses from 2, $F_{i,1}$ is satisfied by a and thus a' satisfies $F'_{i,1}$. If there is an $i^* \in [i]$ such that all $p_{i^*,j}$ with $j \in [2, n+1]$ take the value 0, then set $a(p_{i^*,1}) := 1$ and $a(p_{k,1}) := 0$ for all other $k \in [i], k \neq i^*$. As before, all clauses of $F_{i,1}$ are satisfied and thus $F'_{i,1}$ is satisfied by a'.

For the other direction, assume that a' satisfies $F'_{i,1}$ and that a is an extension of a' that satisfies $F_{i,1}$. If there is an $i^* \in [i]$ such that all $p_{i^*,j}$ with $j \in [2, n+1]$ take the value 0, then there is nothing to show. So assume that for every $k \in [i]$ there is a $j' \in [2, n+1]$ such that $a'(p_{k,j'}) = 1$. Since $F_{i,1}$ contains the clause $\bar{p}_{k,1} \vee \bar{p}_{k,j'}$, we have $a(p_{k,1}) = 0$. Since this is true for all $k \in [i]$ and a satisfies ALO_1, there must be $i' \in [i+1, n]$ which is set to 1 by a' which completes the proof of the claim. ◁

It follows that $F_{i,1}$ can be expressed as a small BDD with variable order π_r: check for every fixed $i' \in [i]$ if the value of all $p_{i,j}$ is 0. Since, for every $i \in [n]$, these variables are consecutive in π_r, this can be easily done by a BDD that is essentially a path. We then glue these BDDs in increasing order of i' in the obvious way and check for $i' \in [i+1, n]$ if $p_{i',1}$ takes value 1 to get a BDD for $F_{i,1}$ of size $O(n^2)$, since we have to consider $O(n^2)$ variables.

We now consider the case $j > 2$. In that case, $F_{i,j}$ contains all variables of PHP_n. Note that if $j = n+1$, then $F_{i,j} = \mathrm{PHP}_n$ and thus the formula $F'_{i,j}$ is unsatisfiable and has a constant size encoding as a BDD. So assume in the remainder that $j \leq n$. Consider an assignment a' to $F'_{i,j}$. Let $I_{a'}$ be the set of indices $i' \in [n]$ such that for all j' for which $p_{i',j'}$ appears in $F'_{i,j}$ we have $a'(p_{i',j'}) = 0$.

▷ Claim 3. a' satisfies $F'_{i,j}$ if and only if

1. $|I_{a'}| \geq j$ and there is an index $i' \in I_{a'} \cap [i]$, or
2. $|I_{a'}| \geq j-1$ and there is an $i^* \in [i+1, n] \setminus I_{a'}$ such that $a'(p_{i^*,j}) = 1$.

Proof. Let first Case 1 be true. Then we can construct a injective function $f : [j] \to I_{a'}$ with $f(j) \in I_{a'} \cap [i]$. We construct an extension a of a' to all variables of $F_{i,j}$ as follows: for $j' \in [j]$ we set $a(p_{f(j'),j'}) := 1$ and set all other variables to 0. Then for $j' \in [j]$, the clause $\mathrm{ALO}_{j'}$ is satisfied by $p_{f(j'),j'}$. Let $V_{i,j}$ the variables of a not assigned in a'. For every $i' \notin I_{a'}$, the variables $p_{i',k} \in V_{i,j}$ are assigned to 0, so all clauses of the form $\bar{p}_{i',k} \vee \bar{p}_{i',\ell}$ in $F_{i,j}$ are satisfied. If $i' \in I_{a'}$, then, since f is injective, at most one variable $p_{i',k}$ with $k \in [n+1]$ is assigned to 1, so a satisfies AMO_i and thus in particular all clauses of the form $\bar{p}_{i',k} \vee \bar{p}_{i',\ell}$. Thus, a satisfies $F_{i,j}$ and a' satisfies $F'_{i,j}$.

Now assume Case 2 is true. Construct a injective function $f : [j-1] \to I_{a'}$. We again construct an extension a of a': for $j' \in [j-1]$ we set $a(p_{f(j'),j'}) := 1$ and set all other variables in $V_{i,j}$ to 0. We show that all clauses of $F_{i,j}$ are satisfied by a. First, ALO_j is satisfied by $p_{i^*,j}$. For $j' \in [j-1]$, the clause ALO_j is satisfied by $p_{f(j'),j'}$. For the binary clauses, we reason exactly as in the previous case. It follows that a' satisfies $F_{i,j}$.

For the other direction, assume that a' satisfies $F'_{i,j}$ and let a be an extension of a' that satisfies $F_{i,j}$. Since a must in particular satisfy the clauses $\mathrm{ALO}_{j'}$ for $j' \in [j-1]$, we can choose, for every $j' \in [j-1]$, an index $f(j') \in [n]$ such that $a(p_{f(j'),j'}) = 1$. All variables in the clauses ALO_k for $k \in [j-1]$ are in $V_{i,j}$, so all binary clauses $p_{i',k} \vee \bar{p}_{i',\ell}$ with $k \in [j-1]$ appear in $F_{i,j}$. So in particular, there cannot be $k \in [j-1], \ell \in [n+1]$ such that $a(p_{f(k),k}) = a(p_{f(k),\ell}) = 1$. It follows that f is injective and for every $k \in [j-1]$ we have that $f(k) \in I_{a'}$. It follows that $|I_{a'}| \geq j-1$.

Now assume that Case 1 is false. Say first that $|I_{a'}| \ngeq j$, so $|I_{a'}| = j-1$. Then f is a bijection. The clause ALO_j is satisfied by a, so there must be $i^* \in [n]$ such that $a(p_{i^*,j}) = 1$. We claim that i^* cannot be in $I_{a'}$. By way of contradiction, assume this were wrong. Then, because f is a bijection, there is $j' \in [j-1]$ with $f(j') = i^*$. By construction of f, we have $a(p_{f(j'),j'}) = a(p_{i^*,j'}) = 1$. Then, because $F_{i,j}$ contains the clause $\bar{p}_{i^*,j'} \vee \bar{p}_{i^*,j}$, the assignment a does not satisfy $F_{i,j}$ which is a contradiction. So $i^* \notin I_{a'}$. Moreover, $i^* > i$ since otherwise all binary clauses $\bar{p}_{i^*,j} \vee \bar{p}_{i^*,\ell}$ would be in $F_{i,j}$ and thus i^* would be in $I_{a'}$. So in this case we have that Case 2 is true.

If there is no index $i' \in I_{a'} \cap [i]$, then we claim that ALO_j is satisfied by a variable $p_{i^*,j}$ for $i^* > i$: reasoning with the binary clauses similarly to before, whenever $a(p_{i',j}) = 1$ for some $i' \in [i]$, then $i' \in I_{a'}$. So none of the $p_{i',j}$ with $i' \in [i]$ satisfy ALO_j and it is satisfied by some $p_{i^*,j}$ which appears in $F'_{i,j}$. Then $i^* \notin I_{a'}$ due to $a'(p_{i^*,j}) = 1$, so Case 2 is true. ◁

With Claim 3, we can bound the size of the BDD encoding $F'_{i,j}$: since for every $i' \in [n]$ the variables $p_{i',j'}$ are consecutive in the order π_r, we can check if $i' \in I_{a'}$ by a BDD of constant width. By making j parallel copies of this BDD for every i', we can compute the size of $I_{a'}$ cutting off at j in width $O(j)$. We can also check if there is an index $i' \in I_{a'} \cap [i]$ or $i^* \in [i+1, n] \setminus I_{a'}$ such that $a'(p_{i^*,j}) = 1$ with only a constant additional factor. Since $F'_{i,j}$ has $O(n(n-j))$ variables, the overall size of the BDD computing $F'_{i,j}$ is $O(j(n-j)n) = O(n^3)$.

It remains to bound the size of BDDs we get from the conjoin-steps between quantification steps. So consider a conjoin step before quantifying $p_{i,j}$ but after the potential previous quantification. Call the resulting BDD D and let D' be the BDD we got from the previous quantification (if there is no previous quantification, set D' to the constant 1 BDD).

▷ Claim 4. The size of D is $O(n^3)$.

Proof. We first claim that when we start conjoining the BDDs in the bucket of variable $p_{i,j}$, the only BDD that does not encode a clause is D'. This is because after every quantification step the result contains the next variable in the order π_c. Thus, since we conjoin only BDDs that contain the variable $p_{i,j}$, the BDDs involved in these steps are D' and potentially BDD representations of ALO_j and clauses $\bar{p}_{i,j} \vee \bar{p}_{i,k}$ for $k > j$. First assume that the conjunction only involves clauses $\bar{p}_{i,j} \vee \bar{p}_{i,k}$ for $k > j$. We claim that the result then has size $O(d)$ where d is the number of conjuncts involved. To see this, observe that if $p_{i,j}$ takes value 0, then all clauses in the conjunction are true, so the conjunction evaluates to true as well so we can directly go to the 1-sink. If $p_{i,j}$ takes value 1, then we have to verify if all other $p_{i,k}$ involved in the conjunction are 0 which can be done by a path of length d because $p_{i,j}$ is the first variable in π_r.

If the conjunction also involves ALO_j, then if $p_{i,j}$ takes value 1, we proceed as before since ALO_j is satisfied already. For the case where $p_{i,j}$ takes value 0, we have to check all other variables in ALO_j on a path. ALO_j is only conjoined if $i = 1$, so in that case $p_{i,j}$ is again the first variable to consider, so this procedure can be done following the order π_r. Overall, the conjunction in this case has size $O(n)$. Note also that in all cases discussed so far, we can also represent the conjunction by a BDD of constant width.

It remains to consider the case in which D' is involved in the conjunctions. We can then see the conjunction as one of several clauses, as discussed above, and D'. As shown above, D' has width $O(j) = O(n)$, so this is also true for the result D of conjoining some of the clauses, since the latter contribute only constant width. So D has a BDD of size $O(n^3)$. ◁

We have shown that all BDDs that we ever construct in the refutation have size at most $O(n^3)$. We make $O(n^3)$ conjoin operations and $O(n^2)$ quantifications and all BDD-operations can be performed in time polynomial in the input, so the overall runtime is polynomial. ◀

4 No Closure Under Restrictions

We next show that Theorem 1 is not true for restrictions of PHP_n. To this end, we consider the graph $G = ([2n], [2n+1], E)$ where the edge set E is defined by

$$E = \{(j,j), (n+j,j), (j,n+1+j), (n+j,n+1+j), (j,n+1), (n+j,n+1), \mid j \in [n]\}.$$

▶ **Theorem 5.** *Bucket elimination in which all BDDs have order π_r and the elimination proceeds in order π_c refutes G-PHP in time $\Omega(2^n)$.*

Proof. We will show that bucket elimination constructs an exponential size BDD in its run. To this end, we first give all the clauses of G-PHP (with the constraint names below):

$$\begin{aligned}
&\bigwedge_{j\in[n]} \underbrace{\left((\bar p_{j,j} \vee \bar p_{j,n+1+j}) \wedge (\bar p_{j,j} \vee \bar p_{j,n+1}) \wedge (\bar p_{j,n+1} \vee \bar p_{j,n+1+j})\right)}_{\mathrm{AMO}_j}\\
\wedge &\bigwedge_{j\in[n]} \underbrace{\left((\bar p_{n+j,j} \vee \bar p_{n+j,n+1+j}) \wedge (\bar p_{n+j,j} \vee \bar p_{n+j,n+1}) \wedge (\bar p_{n+j,n+1} \vee \bar p_{n+j,n+1+j})\right)}_{\mathrm{AMO}_{n+j}}\\
\wedge &\bigwedge_{j\in[n]} \Big(\underbrace{(p_{j,j} \vee p_{n+j,j})}_{\mathrm{ALO}_j} \wedge \underbrace{(p_{j,n+1+j} \vee p_{n+j,n+1+j})}_{\mathrm{ALO}_{n+1+j}}\Big) \wedge \underbrace{\bigvee_{j\in[n]} p_{j,n+1} \vee p_{n+j,n+1}}_{\mathrm{ALO}_{n+1}}
\end{aligned}$$

We consider the step directly before the quantification of $p_{2n,n+1}$, so after conjoining the contents of $B_{p_{2n,n+1}}$ to a BDD D in Line 6 in Algorithm 1. We claim that D has exponential size. To this end, first observe that, at the time of the construction of D, all clauses have been joined except ALO_{n+1+j} which contain no variables $p_{i,j'}$ with $j' \in [n+1]$. We claim that all these clauses have contributed to D. Indeed, whenever eliminating $p_{i,j}$ with $j \in [n]$, the result contains the variable $p_{i,n+1}$ and will thus be put into the bucket $B_{p_{i,n+1}}$ eventually. Then the clause ALO_{n+1} makes sure that all these BDDs are (after some more conjoining and quantification) contributing to D. So we get D by conjoining all clauses except the ALO_{n+1+j} and eliminating all variables up to $p_{2n-1,n+1}$.

Let F be the function we get from D by fixing $p_{n,2n+1}, p_{2n,2n+1}$ to 0 and $p_{2n,n+1}$ to 1. Let F' be the corresponding conjunction of clauses. Then ALO_{n+1} is satisfied and the remaining literals $\bar p_{i,n+1}$ are all pure in F'. Thus, by pure variable elimination, an assignment a to the variables $p_{j,n+1+j}, p_{n+j,n+1+j}$ for $j \in [n-1]$, which are the variables of F, can be extended to a satisfying assignment of F' if and only if it can be extended to a satisfying assignment of

$$\bar p_{2n,n} \wedge (p_{n,n} \vee p_{2n,n}) \wedge \bigwedge_{j\in[n-1]} (\bar p_{j,j} \vee \bar p_{j,n+1+j}) \wedge (\bar p_{n+j,j} \vee \bar p_{n+j,n+1+j}) \wedge (p_{j,j} \vee p_{n+j,j}).$$

Eliminating $p_{j,j}, p_{n+j,j}$ for $j \in [n]$, we see that F is equivalent to

$$\bigwedge_{j\in[n-1]} (\bar p_{j,n+1+j} \vee \bar p_{n+j,n+1+j}).$$

When representing F in a BDD with row-wise variable order, all $p_{j,n+1+j}$ are before all $p_{n+j,n+1+j}$, so we are, up to renaming literals which does not change the size of a BDD, in the situation of Lemma 1. We get that any BDD for F with the order π_r has size at least 2^{n-1} and, since fixing variables does not increase the size of BDD-representations, we get the same lower bound for D. ◀

5 Lower Bounds for Single Orders

We will now analyze bucket elimination in which the elimination order is also the order in which variables appear in the BDDs. This is the behavior of the implementations of [15, 14]; the implementation of [6] allows the use of two orders in the current version. In the variant with one order, which we call *single-order bucket elimination*, it is always the variable in the source of the BDDs that is eliminated, which makes the algorithm simpler. We show here that neither of the two orders introduced in Section 3 leads to polynomial runtime behavior on its own, suggesting that it is the combination of both that is required for efficiency.

▶ **Lemma 2.** *Single-order bucket elimination for PHP_n with order π_c constructs an intermediate BDD of size 2^n.*

Proof. Consider the situation after we have eliminated the variables $p_{1,1}, p_{2,1}, \dots, p_{n,1}$. As analyzed in the proof of Theorem 1, after eliminating the last of these variables, we have constructed a BDD for the function $F'_{n,1}$ that is satisfied by an assignment if and only if there is an i such that all $p_{i,j}$ with $j \in [2, n+1]$ take the value 0. We claim that the BDD-representation of $F'_{n,1}$ has exponential size.

To show this, we consider the restriction F of $F'_{n,1}$ that we get by fixing all variables $p_{i,j}$ for $j > 3$ to 0. Thus, F has the variables $p_{1,2}, p_{1,3}, p_{2,2}, p_{2,3}, \dots, p_{n,2}, p_{n,3}$. We rename for all $i \in [n]$ the variables $p_{i,2}$ to x_i and $p_{i,3}$ to y_i. The resulting function F' evaluates to 1 if and only if there is an i such that x_i and y_i take the value 0. Then the negation $\bar{F}$ of F is given by $\bar{F} = \bigwedge_{i\in[n]} x_i \vee y_i$. Moreover, the variable order of the BDD we have to consider has all x_i before any y_i, and thus, by Lemma 1, any BDD for $\bar{F}$ has size at least 2^n. Since BDDs allow negation and restrictions without size increase, this shows the lower bound for $F_{n,1}$ and thus the claim. ◀

▶ **Lemma 3.** *Single-order bucket elimination for PHP_n with order π_r constructs an intermediate BDD of size 2^n.*

Proof. Consider the BDD D we construct after eliminating the first row. The clauses that contribute to this function are all clauses of AMO_1 as well as all ALO_j for all $j \in [n+1]$. When quantifying away the $p_{1,j}$ for $j \in [n]$, we get a function F that is satisfied by an assignment a if there is a $j^* \in [n+1]$ such that for all $j \in [n+1] \setminus \{j^*\}$ there is a $p_{i,j}$ set to 1 by a; this is because all ALO_j have to be satisfied and at most one of them can be satisfied by $p_{1,j}$ due to AMO_1. F has to be represented by a BDD in bucket elimination and we will show that this requires exponential size. To see this, fix all variables $p_{i,j}$ for $i > 3$ to 0 and fix $p_{2,n+1}$ and $p_{3,n+1}$ to 0. The resulting function is $F' = \bigwedge_{j\in[n]} p_{2,j} \vee p_{3,j}$. In the BDD-representation, all $p_{2,j}$ come before any $p_{3,j}$, so, up to renaming the variables $p_{2,j}$ to x_j and $p_{3,j}$ to y_j, we are in the situation of Lemma 1 and, observing that fixing variables does not increase the size of a BDD, the lower bound follows from there. ◀

6 Conclusion

We have shown that bucket elimination based SAT-solving using BDDs can efficiently solve pigeonhole principle formulas, theoretically confirming prior experimental work from [9, 10, 6], which worked with a slightly different encoding. We have also seen that this result is not stable under restrictions, showing that, at least for the same orders, there are formulas we get by restriction of the pigeonhole principle that take exponential time to solve. We have also seen that the common single-order variant of bucket elimination [15, 14, 6] has exponential runtime for the two variable orders that in combination allow efficient solving.

For practical SAT-solving with BDD-based solvers, our results are mixed news: while we confirm that these solvers are in principle powerful in the sense that they can efficiently solve instances that are out of reach for resolution and thus CDCL-solvers, our additional results suggest that in general it might be hard to come up with the right two variable orders for the instances at hand, in particular since orders good for one type of formulas are bad for very related formulas. So it is not clear how useful BDD-based bucket elimination will be beyond very restricted formula classes.

We close the paper with some questions. First, it would be interesting to understand if for every bipartite graph G the formula G-PHP can be refuted efficiently by choosing orders adapted to the problem or if there are graphs for which bucket elimination is slow for all order choices. In particular, one might also consider some of the many different variants of the pigeonhole principle or mutilated chessboard formulas which have been considered extensively in the literature as benchmarks for solvers but also in theoretical work, see e.g. the overview in [16].

Finally, it is not clear if single-order bucket elimination can solve PHP_n efficiently for some order. The experimental work in [5, 10] does not show any such order, and our own search in this direction has shown only lower bounds that are variants of those presented in Section 5. It is thus natural to conjecture that in fact single-order variable bucket elimination cannot solve PHP_n efficiently. Note that proving this would in particular show that two orders make the approach strictly more powerful.

References

1 Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes and width parameters. *Theory Comput. Syst.*, 64(5):861–914, 2020. doi:10.1007/s00224-019-09930-2.

2 Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In *10th International Conference Principles and Practice of Constraint Programming,CP 2004,*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004. doi:10.1007/978-3-540-30201-8_9.

3 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48(2):149–169, 2001. doi:10.1145/375827.375835.

4 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. doi:10.3233/FAIA336.

5 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

6 Randal E. Bryant and Marijn J. H. Heule. Generating extended resolution proofs with a bdd-based SAT solver. In *27th International Conference Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2021*, volume 12651 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 2021. doi:10.1007/978-3-030-72016-2_5.

7 Sam Buss, Dmitry Itsykson, Alexander Knop, Artur Riazanov, and Dmitry Sokolov. Lower bounds on OBDD proofs with several orders. *ACM Trans. Comput. Log.*, 22(4):26:1–26:30, 2021. doi:10.1145/3468855.

8 Wei Chen and Wenhui Zhang. A direct construction of polynomial-size OBDD proof of pigeon hole problem. *Inf. Process. Lett.*, 109(10):472–477, 2009. doi:10.1016/j.ipl.2009.01.006.

9 Cayden R. Codel, Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant. Bipartite perfect matching benchmarks. http://www.pragmaticsofsat.org/2021/BiPartGen-slides.pdf, 2021. presentation at Pragmatics of SAT 2021 workshop.

10 Cayden R. Codel, Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant. Bipartite perfect matching benchmarks. unpublished report available at https://www.cs.cmu.edu/~jereeves/research/bipart-paper.pdf, 2021.

11 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM (JACM)*, 7(3):201–215, 1960.

12 Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999. doi:10.1016/S0004-3702(99)00059-4.

13 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

14 Toni Jussila, Carsten Sinz, and Armin Biere. Extended resolution proofs for symbolic SAT solving with quantification. In *International Conference on Theory and Applications of Satisfiability Testing, SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 54–60. Springer, 2006. doi:10.1007/11814948_8.

15 Guoqiang Pan and Moshe Y. Vardi. Symbolic techniques in satisfiability solving. *J. Autom. Reason.*, 35(1-3):25–50, 2005. doi:10.1007/s10817-005-9009-7.

16 Alexander A. Razborov. Resolution lower bounds for perfect matching principles. *J. Comput. Syst. Sci.*, 69(1):3–27, 2004. doi:10.1016/j.jcss.2004.01.004.

17 Irina Rish and Rina Dechter. Resolution versus search: Two strategies for SAT. *J. Autom. Reason.*, 24(1/2):225–275, 2000. doi:10.1023/A:1006303512524.

A Proofs for Section 2 (Preliminaries)

▶ **Lemma 1.** *Every BDD computing $\bigwedge_{i\in[n]} x_i \vee y_i$ with a variable order in which every x_i comes before every y_j has at least 2^n nodes.*

Proof. Consider the partition (X, Y) where X contains all the x_i and Y all the y_i. We start by presenting a known connection to so-called rectangle covers. A *rectangle* respecting the partition (X, Y) is a function $f(X, Y)$ that can be written as a conjunction

$$f(X, Y) := f_1(X) \wedge f_2(Y).$$

All rectangles we consider here will respect (X, Y), so we do not mention it here explicitly in the remainder. We say that an assignment a *lies in the rectangle* f, if $f(a) = 1$. A *rectangle cover* of a function $f(X, Y)$ is a sequence $f^1, \ldots, f^s$ such that

$$f(X, Y) = \bigvee_{i\in[s]} f^i(X, Y) \tag{1}$$

where the f^i are all rectangles. The size of the rectangle cover is defined to be s. Rectangles are connected to BDDs due to the following result:

▷ Claim 6. If a function f can be represented by a BDD of size s with a variable order in which all variables in X appear before those in Y, then there is a rectangle cover of f of size s.

Proof. Let D be a BDD computing f. Let $v_1, \ldots, v_\ell$ the nodes with a label not in X such that there is an edge from a node with label in X to v_i. Remember that every assignment a to (X, Y) induces a path through D. Let for every $i \in [\ell]$ the Boolean function f^i be defined as the function accepting exactly the assignments a accepted by D and whose path leads through a. Then f^i is a rectangle, since we can freely combine the paths from the source to v_i with those from v_i to the 1-sink. Moreover, every assignment accepted by D must be accepted by at least one v_i since the 1-sink is not labeled by X but the source of D is (ignoring trivial cases here in which f does not depend on X where the statement is clear because f itself is a rectangle cover of itself). So we get that

$$f(X, Y) = \bigvee_{i\in[\ell]} f^i(X, Y)$$

and thus f has a rectangle cover of size ℓ. Observing that D has at least the nodes $v_1, \ldots, v_\ell$ which are all different and thus at least size ℓ, completes the proof of the claim. ◁

We will show a lower bound on the size of any rectangle cover of the function $f(X,Y) := \bigwedge_{i\in[n]} x_i \vee y_i$, using the so-called fooling set method as follows: consider the set M of models of $f(X,Y)$ of Hamming weight exactly n. These models assign for each $i \in [n]$ exactly one of x_i and y_i to 1.

▷ Claim 7. In any rectangle cover of f, no two assignments $a, b \in M$ with $a \neq b$ lie in the same rectangle.

Proof. By way of contradiction, assume that there is a rectangle cover of f such that there are $a, b \in M$ with $a \neq b$ that lie in the same rectangle f^j. Let a_X be the restriction of a to X and a_Y that to Y. Define b_X and b_Y analogously. Since a and b are not equal, there is an $i \in [n]$ where $a(x_i) \neq b(x_i)$ or $a(y_i) \neq b(y_i)$. Since we have by the choice of M that $a(x_i) = \neg a(y_i)$ and similarly $b(x_i) = \neg b(y_i)$, we actually get that $a(x_i) \neq b(x_i)$ and $a(y_i) \neq b(y_i)$ are both true. It follows that $a(x_i) = b(y_i) \neq a(y_i) = b(x_i)$.

Now assume w.l.o.g. that $a(x_i) = b(y_i) = 0$. Then for $c = a_X \cup b_Y$ we have $f(c) = 0$. But since f^j is a rectangle, we have that $f^j(c) = f_1^j(a_X) \wedge f^j(b_Y) = 1$ which is a contradiction. So a and b cannot lie in the same rectangle, as claimed. ◁

Note that for every assignment a_X to X there is an extension to Y such that the resulting assignment a is in M (simply set for every $i \in [n]$ the missing value by $a(y_i) := \neg a(x_i)$). Thus, M has size 2^n. By Claim 7 and Claim 6, we get that any rectangle cover of f and thus any BDD for f has size 2^n which completes the proof. ◀

Solving Huge Instances with Intel® SAT Solver

Alexander Nadel
Intel Corporation, Haifa, Israel
Faculty of Data and Decision Sciences, Technion, Haifa, Israel

Abstract

We introduce a new release of our SAT solver `Intel® SAT Solver`. The new release, called `IS23`, is targeted to solve huge instances beyond the capacity of other solvers. `IS23` can use 64-bit clause-indices and store clauses compressedly using bit-arrays, where each literal is normally allocated fewer than 32 bits. As a preliminary result, we show that only `IS23` can handle a gigantic trivially satisfiable instance with over 8.5 billion clauses. Then, we demonstrate that `IS23` enables a significant improvement in the capacity of our industrial tool for cell placement in physical design, since only `IS23` can solve placement instances with up to 4.3 billion clauses. Finally, we show that `IS23` is substantially more efficient than other solvers for finding many (10^6) placements on instances with up to 170 million clauses. We use the latter application to demonstrate that *variable succession*, that is, the order in which the variables are provided to the solver, might have a significant impact on `IS23`'s performance, thereby providing a new dimension to SAT encoding considerations.

2012 ACM Subject Classification Mathematics of computing → Solvers

Keywords and phrases SAT, CDCL

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.17

Supplementary Material *Software (Source Code)*: `https://github.com/alexander-nadel/intel_sat_solver`, archived at `swh:1:dir:f665615bfa3f6d0af72cf9e1b4ec027c57a9f4c7`

1 Introduction

A SAT solver decides the classical NP-complete problem of whether the given propositional formula F in Conjunctive Normal Form (CNF)[1] is satisfiable. Modern Conflict-Driven-Clause-Learning (CDCL) SAT solvers are widely used [5]. They implement backtrack search, enhanced by *conflict clause* learning and many other techniques.

SAT research is mostly focused on developing algorithms for solving, within the given timeout, empirically difficult, but not necessarily large benchmarks. In this study, we targeted improving the SAT capacity to enable solving huge instances with billions of clauses (cf. the size of the instances in the main track of the latest SAT competition 2022 [2] ranged from 264 to 214,309,011 clauses with 7,117,471 being the average). SAT solvers might fail on huge instances due to limitations related to memory management, uncharacteristic for other use-cases. To better understand these limitations, recall how SAT solvers manage clauses.

Long clauses (that is, clauses having at least 3 literals) are stored in the so-called *clause buffer*. An initial clause C is typically represented by $(1 + |C|)$ 32-bit words, containing C's size, followed by C's literals (where conflict clauses have some extra-fields). Let *variable succession* be the order in which the variables are provided to the solver. The internal indices, which represent variables and literals, depend on the variable succession. In most solvers since `MiniSat` [9], a positive literal v_i (where i reflects its order in the succession) is represented by the 32-bit *literal-index* $li(v_i) = 2 \times i$, while a negative literal $\neg v_i$ is represented by $li(\neg v_i) = 2 \times i + 1$. For example, consider the following formula E:

$$E = (C_1 = v_1 \vee v_2 \vee \neg v_3) \wedge (C_2 = \neg v_1 \vee v_2 \vee v_3)$$

[1] A CNF formula is a conjunction of clauses (aka, *initial clauses*), each *clause* being a disjunction of Boolean literals, where a *literal* is a Boolean variable or its negation.

E would be represented in the clause buffer by the following eight 32-bit words:

$$\left\langle \overbrace{\;3\;}^{|C_1|}, \overbrace{2,4,7}^{C_1}, \overbrace{\;3\;}^{|C_2|}, \overbrace{3,4,6}^{C_2} \right\rangle$$

In order to uniquely identify and access a clause C, solvers use its *clause-index* ci(C) in the clause buffer. In our example, $ci(C_1)$=0 and $ci(C_2)$=4.

The fundamental capacity limitation of the older solvers (such as `MiniSat` and `Glucose` [1]), but also some of the most modern solvers (such as `MergeSat` [15] and the baseline solver for recent SAT competition winners `Kissat` [4]) is caused by their clause-index width being limited to 32 bits or even fewer due to additional bookkeeping (e.g., 31 bits in `Kissat`).

The first open-source solver to offer a 64-bit-clause-index version was `CryptoMiniSat` [25], which can be compiled with a 64-bit clause-index since May 2017 [24]. A 64-bit clause-index is also used by `CaDiCaL` [4]. Although using 64-bit clause-indices eliminates the major SAT capacity limitation while not affecting the size of the clause buffer, it comes with the price of inflating data structures which point to clauses (notably, including the Watch Lists (WLs) [17], which contain two clause-indices per clause), thus increasing the solver's memory consumption.

`Intel® SAT Solver` (`IntelSAT`) is our CDCL SAT solver, which we released as open-source last year [18]. We optimized it for incremental SAT solving in the presence of many satisfiable queries. The original `IntelSAT` uses a 32-bit clause-index. This paper introduces a new release of `IntelSAT` – `IS23`, aimed at extending the solver's capacity. `IS23` can be compiled into various versions, including the default `IS23` (similar to the original `IntelSAT`), `IS23-64` and `IS23-64C`. `IS23-64` extends the clause-index width from 32 bits to 64 bits. `IS23-64C` uses *bit-arrays* to store clauses *compressedly*, where the goal is to reduce the memory footprint (thus, potentially, also reducing the number of cache misses) at the expense of applying additional bit-wise operations to access clauses.

We demonstrate our core idea on our example formula $E = (C_1 = v_1 \vee v_2 \vee \neg v_3) \wedge (C_2 = \neg v_1 \vee v_2 \vee v_3)$. Given a clause C, let its *literal-width* $lw(C)$ be the minimal number of bits required to store its highest literal index. To store C, we allocate its every literal $lw(C)$ bits. Observe that, in E, we have $lw(C_1) = lw(C_2) = 3$, thus the formula (without the clause sizes) can be represented using 18 bits as $\left\langle \overbrace{\underbrace{010}_{2}, \underbrace{100}_{4}, \underbrace{111}_{7}}^{C_1}; \overbrace{\underbrace{011}_{3}, \underbrace{100}_{4}, \underbrace{110}_{6}}^{C_2} \right\rangle$ (in binary encoding), which requires only one 32-bit word instead of eight. Apparently, to access clause's literals, the literal-width must be known upfront. To support clauses with arbitrary literal-widths, `IS23-64C` stores clauses in multiple bit-arrays, where all the clauses in a single bit-array share the same literal-width (along with two other fields as detailed in Sect. 3). In `IS23-64C`, the 64-bit clause-index of every clause C contains the unique ID of C's bit-array (11 bits) and the bit number where C starts in its bit-array (the remaining 53 bits).

Notably, the `sharpSAT` model counter [26] first applied the idea of storing subsets of clauses (aka components) compressedly by limiting the number of bits in every clause to the maximal literal-width in that component. However, while `sharpSAT` only stashed the components compressedly for future usage, we have implemented a full-fledged CDCL SAT solver with the compressed clause buffer as the underlying data structure.

We carried out several experiments to evaluate the different versions of `IS23` against other solvers, including `Kissat`, `CaDiCaL`, `CryptoMiniSat` and `MergeSat`.

In our first preliminary experiment, we show that only IS23-64C can solve a huge trivially satisfiable instance having 2^{33} = 8,589,934,592 clauses and 292,057,776,128 literals overall (in all the input clauses).

Our own interest in extending the capacity of SAT stems from our industrial placement application. Cell placement is one of the most important problems in VLSI automation [23]. Its most basic version concerns placing without overlap a set of rectangles on a grid. In [8], we have presented our SAT-based placement tool, which starts with finding one placement and then optimizes it with incremental SAT queries. We initiated the development of IS23, since we had been observing an increasing number of cases where our tool failed to find even the initial placement due to capacity limitations of IntelSAT. Furthermore, recently, we encountered the need to solve another flavor of the placement problem, we call *N-placements*: find a user-given number of different placements (from which promising placements are subsequently selected and might be further optimized). In the rest of paper, we consider the problems of finding 1 or $N > 1$ placements, leaving optimization outside of our scope.

In our second experiment, we show that only with IS23 can we find one placement for huge problems, whose corresponding CNF instances have up to 4.3 billion clauses.

In our third experiment, we show that only IS23-64C can find 1,000,000 placements for instances having up to 170 million clauses, where, to achieve the best results, the variable succession scheme must be carefully chosen.

The rest of this paper is organized as follows. Sect. 2 presents preliminaries. Sect. 3 introduces IS23. Sect. 4 is about experimental results. Sect. 5 concludes our work.

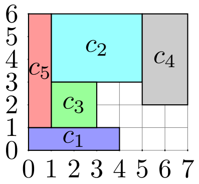
2 Preliminaries

A *literal* l is a Boolean variable v, in which case l is *positive*, or a variable's negation $\neg v$, in which case l is *negative*. A *clause* is a disjunction of literals. Let an *n-clause* and *>n-clause* be a clause of size n and >n, respectively. A *long clause* is a >2-clause; a *binary clause* is a 2-clause.

A formula F is in *Conjunctive Normal Form (CNF)* if it is a conjunction (set) of clauses. A SAT solver receives a CNF formula F and returns a satisfying assignment (aka, model or solution) μ, which assigns a Boolean value $\mu(v) \in \{0, 1\}$ to every variable v, where $\mu(\neg v) = \neg\mu(v)$. For a literal l, let *the projection of l in μ* ${}_{\mu}l \in \{l, \neg l\}$ be l iff $\mu(l) = 1$ or, otherwise, $\neg l$.

In *incremental SAT solving (under assumptions)* [9, 21], the user may invoke the solver multiple times, each time with a new set of zero or more *assumption literals* (called, simply, the *assumptions*), while adding zero or more clauses in-between the queries. The solver then checks the satisfiability of all the clauses provided so far, while enforcing the values of the current assumptions.

Cell placement (placement) is one of the most important problems in VLSI automation [23]. We consider the following basic (but already NP-complete [13]) version which concerns placing without overlap a set of rectangles on a grid. The input of the placement problem comprises the following two components: a *rectangular grid region* of fixed size and a finite set of *rectangular cells* of user-given widths and heights. We are interested in *feasible* placements, that is, placements in which no cell overlaps other cells. An example of a feasible placement is shown below (placing five cells of sizes 4 × 1, 4 × 3, 2 × 2, 2 × 4 and 1 × 5 on a 7 × 6 grid):



To encode placement into SAT, first, we associate two *bitvectors* c^l and c^b with the left-bottom coordinate (c^l, c^b) of every cell c (where a *Bitvector (BV)* $b = \{b_n, b_{n-1}, \dots, b_1\}$ is a sequence of $|b|$ Boolean variables, called *bits*). Second, we create two sets of constraints in BV logic [3] over c^l's and c^b's to ensure that all the cells are inside the grid and there is no overlap. Third, we apply an eager BV solver [10], which, after preprocessing, translates the formula to SAT and solves it using a SAT solver. We refer the reader to [8] for all the details.

This paper also considers the *N-placement* problem of finding a user-given number of placements. To solve N-placement, we apply the following algorithm, we call `SimpleBlock` (first proposed in [16] in the context of model checking). `SimpleBlock`, shown below, iteratively finds a solution (placement) μ and immediately blocks it using a single *blocking clause* containing the falsified literal per every *important variable*, where, in our case, the set of the important variables comprises all the bits of the left-bottom coordinates of every cell:

```
Create a CNF formula $F$ representing the given problem.
Invoke a SAT solver over $F$. Let $\mu$ be the returned model (if any).
while $F$ is satisfied with $\mu$ and the user-given solution threshold $N$ not reached do
    Block the current solution by adding the following blocking clause to $F$:
        $(\bigvee_{c\in\mathcal{C}} \neg{}_{\mu}c^l_1 \vee \neg{}_{\mu}c^l_2 \vee \dots \vee \neg{}_{\mu}c^l_{|c^l|}) \vee (\bigvee_{c\in\mathcal{C}} \neg{}_{\mu}c^b_1 \vee \neg{}_{\mu}c^b_2 \vee \dots \vee \neg{}_{\mu}c^b_{|c^b|})$
    Invoke a SAT solver over $F$. Let $\mu$ be the returned model (if any).
```

To evaluate different SAT solvers within `SimpleBlock`, we have implemented `SimpleBlock` in both `IS23` and `CaDiCaL`, whereas `CryptoMiniSat` already supports it.

AllSAT is the problem of enumerating all the solutions in a CNF formula. In practice, AllSAT tools can stop after finding N solutions, which makes them applicable for solving N-placement. [27] contains an extensive survey of AllSAT approaches; it also presents three state-of-the-art AllSAT tools, called `Toda` *tools (solvers)* herein. The `Toda` tools include one solver per each of the following three families of AllSAT algorithms. The first family of the so-called *blocking solvers* use `SimpleBlock` enhanced (mainly by generalizing each solution by turning as many variables as possible into don't cares, thus shortening the blocking clauses). The second family of *nonblocking solvers* [11] modifies the SAT solver to enumerate the solutions explicitly without using blocking clauses. The third family is based on BDD caching [12] and can be combined with the other two methods. Our empirical evaluation of *N-placement* approaches in Sect. 4.3 includes the `Toda` tools.

3 `IS23`: the New Release of `IntelSAT`

This section introduces the `IS23` release of `IntelSAT`. Sect. 3.1 describes the new parametrized API. Sect. 3.2 is about clause compression.

We would also like to mention a new feature of *out-of-memory recovery*: when the operating system refuses to allocate memory, `IS23` compacts its data structures and retries, rather than immediately returning a failure.

3.1 The API

The users of the solver's C++ library class, denoted herein by IS23 $\langle\alpha, \beta, \gamma\rangle$, can now parametrize the solver at compile-time with the following template parameters:

1. *clause-index width* α: the width of the C++ variables, used to represent the clause indices.
2. *literal-index width* β: the width of the C++ variables, used to represent the literal indices.
3. *compression flag* γ: a Boolean flag indicating whether to *compress* clauses using bit-arrays.

For the solver to compile, α and β must be powers of 2 and the following assertion must hold: $8 \leqslant \beta \leqslant \alpha \leqslant K$ for a K-bit operating system.

The default version is IS23 $\equiv$ IS23 $\langle 32, 32, 0\rangle$. In this paper, we also experiment with IS23-64 $\equiv$ IS23 $\langle 64, 32, 0\rangle$ and IS23-64C $\equiv$ IS23 $\langle 64, 32, 1\rangle$, where IS23-64 and IS23-64C can also be accessed from the command-line of the solver's executable (the executable works with the standard DIMACS file format).

The literal-index width β had been 32 bits for every open-source SAT solver so far, hence they can accommodate *at most* $2^{31} - 1$ variables. In fact, it is $2^{31} - 1$ for CaDiCaL, but only $2^{28} - 1$ for Kissat and CryptoMiniSat due to additional bookkeeping. Specifically, Kissat borrows bits from the literal-index to be able to inline binary clauses (that is, store them in the WLs only without maintaining a copy in the clause buffer), while efficiently implementing inprocessing [6] as well as failed literal probing and vivification [14]. In IntelSAT, the WLs are organized similarly to Kissat, but there is currently no need to borrow bits from the literal-index as inprocessing, failed literal probing and vivification are expected to be too heavy for both solving rapid satisfiable incremental queries (the original IntelSAT application) and solving gigantic instances (the current IntelSAT application).

Notably, IS23 is the first solver which can be compiled to allow for a practically unlimited number of variables ($2^{63} - 1 = 9{,}223{,}372{,}036{,}854{,}775{,}807$ variables using $\beta = 64$, if no bits are borrowed from the literal-index), where borrowing several bits, if required, is not expected to limit the number of variables in practice. One could also potentially take advantage of IS23's architecture for *saving the memory* when the number of variables is limited by $2^{15} - 1$ (using $\beta = 16$). We leave experiments with different literal-index widths to future work.

3.2 Clause Compression

In this section, we describe how IS23-64C manages clauses. For simplicity, we assume herein that the literal-index width β is 32. Similarly to most SAT solvers, IS23 represents a positive literal v_i by the literal index $li(v_i) = 2 \times i$ and a negative literal $\neg v_i$ by the literal index $li(\neg v_i) = 2 \times i + 1$. As we have mentioned, IS23 inlines any binary clauses into the WLs [4, 7], hence the discussion below concerns long clauses only.

For our purposes, a *bit-array* is a data structure which supports reading and writing of up to 64 bits starting from a specific bit to a dynamically allocated buffer (using several bitwise operations for every access [22]). We have engineered efficient bit-array support in IS23.

Recall from Sect. 1 that the literal-width $lw(C)$ represents the minimal number of bits, required to store C's highest literal-index. Our core idea is compressing memory by storing clauses as bit-arrays, where each literal is represented by $lw(C)$ bits, and the width of the clause-size field is also clause-dependent. Consequently, we have implemented a new data structure for storing and accessing clauses, which serves as an alternative for the clause buffer. The vast majority of the solver's code is agnostic to how clauses are managed underneath.

Clearly, to access literals in a clause C, $lw(C)$ must be known. To avoid the overhead of storing $lw(C)$ with every C, we maintain a hash-table of bit-arrays which store clauses, where the bit-array of a given clause C is determined by its 11-bit *hash ID* $hash(C)$, including:

1. 5 bits: the literal-width *lw*,
2. 5 bits: *clause-size-width sw*, that is, the number of bits allocated per clause size, and
3. 1 bit: *learnt-status ls*, that is, whether the clause is learnt or initial.

The last two fields are useful for compactly storing the clause sizes and simplifying the implementation of clause deletion strategies, respectively.

For a clause C, $hash(C)$ is maintained as part of its clause-index ci(C), which, for α=64, leaves more than enough bits (64-11=53) to store the *bit-index*, where the clause starts in its bit-array.

Given C, let $|C|^*$ be *C's compressed size*, which we store instead of C's actual size to save memory (details will follow).

The layout of a clause $C = l_1 \vee l_2 \vee \ldots \vee l_{|C|}$ in a bit-array looks as follows (the width is shown over-brace; `glue`, `stay` and `act` are commonly used fields [1, 18] present only in learnt clauses):

$$\left\langle \overbrace{|C|^*}^{sw(C)}, \underbrace{\overbrace{\texttt{glue}}^{11}, \overbrace{\texttt{stay}}^{1}, \overbrace{\texttt{act}}^{31}}_{\text{learnt clauses only}}; \overbrace{li(l_1)}^{lw(C)}; \overbrace{li(l_2)}^{lw(C)}; \ldots; \overbrace{li(l_{|C|})}^{lw(C)} \right\rangle$$

Given a clause C, how do we determine its clause-size-width $sw(C)$ and its compressed size $|C|^*$? Our guiding principle is to use as few bits as possible. Specifically, we use $sw(C) = 0$ for storing 3-clauses, that is, $|C|^*$ is not stored for them at all. For every $sw(C) > 0$, we reserve the special value $|C|^* = 0$ for clause-deletion heuristic's machinery. Therefore, the clause-size-width $sw(C) = 1$ can accommodate only clauses of size 4, where $|C|^* = 1$ for every such clause. To determine $sw(C)$ for arbitrary clauses, we pre-compute, for clause-size-widths $0 \leqslant w < 32$, the *minimal clause size* $mcs(w)$ stored using w bits to accommodate the special value 0 and as many clauses sizes as possible for every w. The first 10 values and the recursive function for $mcs(w)$ are shown below:

w	0	1	2	3	4	5	6	7	8	9	an arbitrary $n > 2$
$mcs(w)$	3	4	5	8	15	30	61	124	251	506	$mcs(n-1) + 2^{n-1} - 1$

Given a clause C, let w_C be the highest w, such that $|C| \geqslant mcs(w)$. We set $sw(C) = w_C$ and for $|C| > 3 : |C|^* = |C| - mcs(w_C) + 1$.

Let G be the following example formula, where C_1 and C_2 are initial and C_3 is learnt:

$$G = \underbrace{(C_1 = v_1 \vee v_2 \vee \neg v_6) \wedge (C_2 = \neg v_1 \vee v_2 \vee v_6)}_{\text{initial clauses}} \wedge \underbrace{(C_3 = v_1 \vee v_2 \vee v_3 \vee v_4 \vee v_5)}_{\text{learnt clause}}$$

Note that the clauses C_1 and C_2 share the hash ID $\{lw = 4, sw = 0, ls = 0\}$, while C_3 has the hash ID $\{lw = 4, sw = 2, ls = 1\}$. Thus, G would be stored in two bit-arrays as follows (the widths are shown over-brace, while labels appear under-brace):

Bit-array's Hash ID	Clauses
$\underbrace{\overbrace{4}^{5}}_{lw}, \underbrace{\overbrace{0}^{5}}_{sw}, \underbrace{\overbrace{0}^{1}}_{ls}$	$\underbrace{\overbrace{2}^{4}, \overbrace{4}^{4}, \overbrace{13}^{4}}_{C_1}, \underbrace{\overbrace{3}^{4}, \overbrace{4}^{4}, \overbrace{12}^{4}}_{C_2}$
$\underbrace{\overbrace{4}^{5}}_{lw}, \underbrace{\overbrace{2}^{5}}_{sw}, \underbrace{\overbrace{1}^{1}}_{ls}$	$\underbrace{\overbrace{\lvert C_3\rvert^* = 1}^{2}, \overbrace{\texttt{glue}}^{11}, \overbrace{\texttt{stay}}^{1}, \overbrace{\texttt{act}}^{31}, \overbrace{2}^{4}, \overbrace{4}^{4}, \overbrace{6}^{4}, \overbrace{8}^{4}, \overbrace{10}^{4}}_{C_3}$

The 64-bit clause-indices would be as follows:

$ci(C_1) = 2^{61}$	$ci(C_2) = 2^{61} + 12$	$ci(C_3) = 2^{61} + 2^{55} + 2^{53}$
$\overbrace{\underbrace{4}_{lw}}^{5}, \overbrace{\underbrace{0}_{sw}}^{5}, \overbrace{\underbrace{0}_{ls}}^{1}, \overbrace{\underbrace{0}_{bit\text{-}index}}^{53}$	$\overbrace{\underbrace{4}_{lw}}^{5}, \overbrace{\underbrace{0}_{sw}}^{5}, \overbrace{\underbrace{0}_{ls}}^{1}, \overbrace{\underbrace{12}_{bit\text{-}index}}^{53}$	$\overbrace{\underbrace{4}_{lw}}^{5}, \overbrace{\underbrace{2}_{sw}}^{5}, \overbrace{\underbrace{1}_{ls}}^{1}, \overbrace{\underbrace{0}_{bit\text{-}index}}^{53}$

3.2.1 Clause Compression and Variable Succession

Variables succession has an immediate impact on the memory footprint of `IS23-64C`, since it impacts the literal-widths of clauses.

For example, in our latest toy formula G, swapping v_3 and v_6 would reduce the literal-width of both C_1 and C_2 from 4 to 3 without changing the literal-width of C_3, thus saving 1 bit per every literal in C_1 and C_2 and 6 bits overall.

Our core observation is that, if a variable is likely to appear in many clauses, it is crucial for this variable to appear as early as possible in variable succession. In this work, we suggest relying on the expert user knowledge of the problem to determine a good variable succession. We provide two examples below, one of which is backed up by experimental results later in the paper.

Recall the SAT-based `SimpleBlock` N-placement algorithm, where we add many blocking clauses over the same set of important variables. In Sect. 4, we evaluate two versions of `IS23-64C`: `IS23-64CL` has the important variables *first* in the succession, while `IS23-64CH` has them *last*. Unsurprisingly, `IS23-64CL` turns out to be significantly more efficient.

Furthermore, many applications of incremental SAT solving augment clauses with the so-called *selector variables (selectors)* to be able to enable and disable clauses using assumptions. Normally, selectors appear late in variable succession, since they are created after the rest of the instance, thus they are associated with highest possible indices. We expect that, for certain applications, having the selectors early in the succession would have a substantial positive impact on `IS23-64C`'s performance. We leave testing this hypothesis to future work.

Automating the variable succession, that is, having the solver renumber the variables automatically, while still compressing the clauses efficiently, would not be trivial. In principle, the solver could try to figure out a good variable succession out of existing clauses, when a sufficient number of them is provided by the user, and then renumber the variables and compress the clauses. However, that would require to temporarily store a significant amount of clauses *non-compressedly*, which might ruin the compression's efficiency. To alleviate this problem, one might renumber variables and recompress clauses frequently, but that might have a negative impact on the solver's performance. Hence, automating the variable succession is a non-trivial task, which we leave to future work.

4 Experimental Results

We carried out three sets of experiments. We denote by `CrM-32` and `CrM-64` the versions of `CryptoMiniSat` with a 32- and 64-bit clause-index, respectively. We omit the results of the previous version of `IntelSAT`, since the default `IS23` performs at least equally as well, while our goal is introducing the novel `IS23-64` and `IS23-64C` variants of `IS23`.

We dub solver errors and exceptions as follows: `CIErr` or `VIErr` mean that the clause-index space or the variable-index space, respectively, has been exhausted; `TO` or `MO` stand for a time-out or a memory-out, respectively; `Err` stands for other errors (mostly crashes).

The code and the binaries of all the tools and all the benchmarks are publicly available at [20]. Additionally, `IntelSAT`'s repository [19] has been updated to `IS23`.

Table 1 Solving S(n). Rows represent instances. The first column contains n. Each pair of subsequent columns shows the time in seconds and memory in GB for the corresponding SAT solver.

n	Kissat		CaDiCaL		MergeSat		CrM-64		IS23		IS23-64		IS23-64C	
	T	M	T	M	T	M	T	M	T	M	T	M	T	M
27	171	21	172	26	5070	35	247	30	227	21	228	21	261	10
28	374	42	347	52	CIErr		505	60	CIErr		459	44	508	19
29	CIErr		702	105	CIErr		1254	125	CIErr		975	90	1023	43
30	CIErr		1523	226	CIErr		2468	249	CIErr		1914	184	2096	81
31	CIErr		3348	453	CIErr		6117	515	CIErr		4262	379	4509	199
32	CIErr		8186	784	CIErr		Err		CIErr		9064	774	9723	365
33	CIErr		Err		CIErr		Err		CIErr		MO		20311	678

Table 2 Finding one placement. The first three columns provide the number of rectangles (in hundreds), variables in CNF (in millions) and clauses in CNF (in millions). Each subsequent pair or triplet of columns corresponds to one solver. Each shows, for the corresponding solver, either: (1) the run-time (in hours), the memory usage (in GB) and, optionally, the number of conflicts (in thousands), or (2) the reason for a failure.

$\frac{R}{10^2}$	$\frac{V}{10^6}$	$\frac{C}{10^6}$	IS23		IS23-64			IS23-64C			CrM-64		Kissat		CaDiCaL	
			T	M	T	M	$\frac{CO}{10^3}$	T	M	$\frac{CO}{10^3}$	T	M	T	M	T	M
20	152	682	1.4	41	1.6	61	19	2.2	60	19	1.7	114	1.6	64	4.3	151
25	238	1066	CIErr		4.2	95	26	4.3	93	21	7.0	180	CIErr		23.5	217
30	342	1535	CIErr		5.6	138	31	8.9	136	31	VIErr		CIErr		27.1	349
35	466	2089	CIErr		14.4	190	54	13.8	186	39	VIErr		CIErr		TO	
40	608	2728	CIErr		13.8	245	46	24.0	245	44	VIErr		Err		Err	
45	770	3453	CIErr		21.8	308	55	25.7	306	55	VIErr		Err		Err	
50	950	4263	CIErr		33.3	382	59	TO			VIErr		Err		Err	

4.1 Gigantic Trivially Satisfiable Instances

To compare solvers' capacity, we created a family of trivially satisfiable instances as follows.

First, consider the following family U of trivially unsatisfiable instances: U(n) contains 2^n clauses, where every clause contains a literal for every one of the n variables $v_1, \ldots, v_n$, and all the clauses are different (so, every clause falsifies exactly one potential solution). For example, U(2) $= (\neg v_1 \vee \neg v_2) \wedge (\neg v_1 \vee v_2) \wedge (v_1 \vee \neg v_2) \wedge (v_1 \vee v_2)$.

The trivially satisfiable family S is generated from U by adding a new variable v_{n+1} to every clause. For example, S(2) $= (\neg v_1 \vee \neg v_2 \vee v_3) \wedge (\neg v_1 \vee v_2 \vee v_3) \wedge (v_1 \vee \neg v_2 \vee v_3) \wedge (v_1 \vee v_2 \vee v_3)$.

For the experiments in this and the next subsection (Sect 4.2), we used a machine having 790Mb of memory and an Intel® Xeon® processor of 2.70Ghz CPU frequency. Table 1 compares Kissat, CaDiCaL, MergeSat, CrM-64, IS23, IS23-64 and IS23-64C on S instances without any time or memory limits (CrM-32 failed with CIErr already on S(25)).

IS23-64C is the only solver, which can solve the gigantic instance S(33) having 2^{33} = 8,589,934,592 clauses and $2^{33} \times (33 + 1)$ = 292,057,776,128 literals overall.

Note that IS23-64C consumes around half the memory of IS23-64. Why is the gap so low, given that, for e.g. $n = 32$, each literal takes $32/6 = 5.3$ times fewer bits in the clause buffer (so, seemingly, one could expect IS23-64C to use 5 times rather than 2 times less memory than IS23-64)? Shortly, because of the Watch Lists. WLs occupy the same amount of memory for both IS23-64 and IS23-64C, but, for IS23-64C, they dominate the memory consumption using over 60% of the memory. Thus, compressing the WLs is a promising direction for future work.

4.2 Finding One Placement

We generated publicly available placement instances in CNF as follows. Each instance in the family P(R) corresponds to the problem of placing R rectangles of randomly chosen width and height in the range $[1 - 10]$ on a $10^3 \times 10^3$ grid. The results on these instances roughly correspond to results on industrial instances of similar size, which we, unfortunately, cannot share due to IP restrictions.

For finding one placement, we ran `Kissat`, `CaDiCaL`, `MergeSat`, `CrM-32` `CrM-64`, `IS23`, `IS23-64` and `IS23-64C` with the timeout of 48 hours and the memory limit of 512Gb on instances in $[\mathtt{P}(2000), \mathtt{P}(2500), \ldots, \mathtt{P}(5000)]$ (all the solvers failed on $\mathtt{P}(5500)$). The results are shown in Table 2 (`MergeSat` and `CrM-32` are omitted as they solved none of the instances).

Our new release of `IntelSAT`, `IS23`, is clearly the most scalable solver as `IS23-64` solved even the instance $\mathtt{P}(5000)$ having almost 1 billion variables and 4.3 billion clauses, whereas the next best solver `CaDiCaL` managed to solve only the $\mathtt{P}(3000)$ instance, while being $4.8X$ slower and using $2.5X$ more memory than `IS23-64` for $\mathtt{P}(3000)$.

Compare `IS23-64` with `IS23-64C`. Usually, `IS23-64` outperformed `IS23-64C` in terms of run-time. `IS23-64C` never generated more conflicts than `IS23-64`, but was almost always slower, apparently because of the overhead of the bit-wise operations. Surprisingly, `IS23-64C` was only slightly more efficient than `IS23` in terms of memory consumption. Further analysis showed that `IS23-64C` did compress the clause buffer (e.g., by $1.5X$ for $\mathtt{P}(4500)$), but other data structures (WLs and variable/literal-indexed arrays) dominated the memory usage.

4.3 Finding Many Placements

In our last experiment, we evaluated the different solvers for finding $N = 1{,}000{,}000$ placements. Since finding 10^6 placements is substantially more difficult than only one, we used smaller instances in $[\mathtt{P}(200), \mathtt{P}(300), \mathtt{P}(400), \ldots]$. However, we decided to also limit the resources: we used machines with 32Gb of memory only running Intel® Xeon® processors of 3Ghz CPU frequency and set the timeout to 10 hours.

We ran `CaDiCaL`, `CrM-64`, `IS23`, `IS23-64` and the two versions of `IS23-64C`, `IS23-64CL` and `IS23-64CH` (recall Sect. 3.2.1), within the `SimpleBlock` algorithm. We also launched the `Toda` tools (recall Sect. 2): `bc_minisat_all`, `nbc_minisat_all` and `bdd_minisat_all`. The last instance for which at least one solver succeeded to find 10^6 placements was $\mathtt{P}(1000)$.

The results are shown in Table 3. Observe that only `IS23-64CL` was able to find 10^6 placements for all the instances. Unlike for finding one placement, `IS23-64` consumed significantly more memory than `IS23-64CL`, since the long blocking clauses dominated the memory consumption (the size of every blocking clause for $\mathtt{P}(R)$ is the number of important variables $= 20 \times R$). Observe that the various `IS23` versions managed to squeeze the memory usage into 31Gb for several instances of different complexity due to the out-of-memory recovery feature (recall Sect. 3).

`IS23-64CH` failed on two instances providing evidence that variable succession scheme is crucial. In addition to `IS23-64CL` and `IS23-64CH`, we have also tested `IS23-64CD`: the variable succession scheme, generated by default by our in-house eager SMT solver. `IS23-64CD` was able to solve $\mathtt{P}(900)$, but not $\mathtt{P}(1000)$, hence we upgraded our default to `IS23-64CL`.

Notably, `IntelSAT` scaled substantially better than both `CaDiCaL` and `CrM-64` within `SimpleBlock`. The explanation may be related to the *Incremental Lazy Backtracking (ILB)* principle, implemented already in the original `IntelSAT` [18]. Specifically, before every incremental SAT query, `CaDiCaL` and `CrM-64` backtrack to the global decision level after each model is found, while `IntelSAT` backtracks to the highest possible decision level, where the latest blocking clause halts to be falsified. Note that implementing or disabling ILB in any of the solvers would have no impact on the experiments reported in Table 1 and Table 2, since the benchmarks used in these experiments are not incremental.

Finally, our `IS23-64CL`-based *N-placement* tool scaled much better than the state-of-the-art AllSAT solvers (`Toda` tools), despite us using only the basic `SimpleBlock` algorithm, which can be substantially improved by techniques, inspired by blocking AllSAT solvers.

Table 3 Finding 10^6 placements. The first column in both the sub-tables shows the number of rectangles (in hundreds); the upper table also contains two columns with the number of variables and clauses in CNF (in millions). Each subsequent triplet of columns shows, for one solver: the number of solutions (in thousands), the run-time (in hours) and the memory usage (in GB); in case of a failure, the last two columns per solver show its reason instead.

$\frac{R}{10^2}$	$\frac{V}{10^6}$	$\frac{C}{10^6}$	IS23			IS23-64			IS23-64CL			IS23-64CH		
			$\frac{S}{10^3}$	T	M	$\frac{S}{10^3}$	T	M	$\frac{S}{10^3}$	T	M	$\frac{S}{10^3}$	T	M
3	3	15	1000	0.4	15	1000	0.5	16	1000	0.6	10	1000	0.5	14
4	6	27	829	CIErr		1000	0.7	22	1000	0.9	12	1000	0.8	20
5	10	43	644	CIErr		1000	0.9	28	1000	1.3	17	1000	1.2	29
6	14	61	513	CIErr		1000	1.3	31	1000	1.8	23	1000	1.4	31
7	19	84	438	CIErr		1000	1.6	31	1000	2.2	25	1000	1.8	31
8	24	109	372	CIErr		668	MO		1000	2.7	31	1000	2.4	31
9	31	138	338	CIErr		482	MO		1000	3.2	31	769	MO	
10	38	171	265	CIErr		449	MO		1000	3.6	31	682	MO	

$\frac{R}{10^2}$	CrM-64			CaDiCaL			bc_minisat_all			nbc_minisat_all			bdd_minisat_all		
	$\frac{S}{10^3}$	T	M	$\frac{S}{10^3}$	T	M	$\frac{S}{10^3}$	T	M	$\frac{S}{10^3}$	T	M	$\frac{S}{10^3}$	T	M
3	30	TO		15	TO		19	TO		1000	0.1	1	0	Err	
4	15	TO		8	TO		8	TO		1000	3.4	3	0	Err	
5	10	TO		5	TO		0	TO		0	TO		0	TO	
6	6	TO		3	TO		0	TO		0	TO		0	TO	
7	4	TO		2	TO		0	TO		0	TO		0	TO	
8	3	TO		2	TO		0	TO		0	TO		0	TO	
9	0	Err		1	TO		0	TO		0	TO		0	TO	
10	0	Err		1	TO		0	TO		0	TO		0	TO	

5 Conclusion

We introduced the IS23 release of our SAT solver IntelSAT, targeted to solve huge instances beyond the capacity of other solvers. IS23 can compress the memory by storing clauses in bit-arrays. We showed that only IS23 can solve a gigantic trivially satisfiable instance with over 8.5 billion clauses. IS23 also enabled solving huge instances of the industrial placement problem with up to 4.3 billion clauses. Additionally, IS23 turned out to be substantially more efficient than other solvers for finding 10^6 placements on instances with up to 170 million clauses, where a carefully chosen variable succession scheme enabled the best results.

References

1 Gilles Audemard and Laurent Simon. On the glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. doi:10.1142/S0218213018400018.

2 Tomas Balyo, Marijn J.H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2022: Solver and Benchmark Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2022.

3 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at http://smt-lib.org/.

4 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

5 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

6 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in sat solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 391–435. IOS Press, 2021. doi:10.3233/FAIA200992.

7 Geoffrey Chu, Aaron Harwood, and Peter J. Stuckey. Cache conscious data structures for boolean satisfiability solvers. *J. Satisf. Boolean Model. Comput.*, 6(1-3):99–120, 2009. doi:10.3233/sat190064.

8 Aviad Cohen, Alexander Nadel, and Vadim Ryvchin. Local search with a SAT oracle for combinatorial optimization. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 87–104. Springer, 2021. doi:10.1007/978-3-030-72013-1_5.

9 Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi:10.1007/978-3-540-24605-3_37.

10 Vijay Ganesh and David L. Dill. A decision procedure for bit-vectors and arrays. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 519–531. Springer, 2007. doi:10.1007/978-3-540-73368-3_52.

11 Orna Grumberg, Assaf Schuster, and Avi Yadgar. Memory efficient all-solutions SAT solver and its application for reachability analysis. In Alan J. Hu and Andrew K. Martin, editors, *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings*, volume 3312 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2004. doi:10.1007/978-3-540-30494-4_20.

12 Jinbo Huang and Adnan Darwiche. The language of search. *J. Artif. Intell. Res.*, 29:191–219, 2007. doi:10.1613/jair.2097.

13 Richard Korf, Michael Moffitt, and Martha Pollack. Optimal rectangle packing. *Annals OR*, 179:261–295, September 2010. doi:10.1007/s10479-008-0463-6.

14 Chu-Min Li, Fan Xiao, Mao Luo, Felip Manyà, Zhipeng Lü, and Yu Li. Clause vivification by unit propagation in CDCL SAT solvers. *Artif. Intell.*, 279, 2020. doi:10.1016/j.artint.2019.103197.

15 Norbert Manthey. The mergesat solver. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021 – 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 2021. doi:10.1007/978-3-030-80223-3_27.

16 Kenneth L. McMillan. Applying SAT methods in unbounded symbolic model checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2002. doi:10.1007/3-540-45657-0_19.

17 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. doi:10.1145/378239.379017.

18 Alexander Nadel. Introducing intel(r) SAT solver. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAT.2022.8.

19 Alexander Nadel. Intel® SAT Solver. https://github.com/alexander-nadel/intel_sat_solver, 2022–2023.

20 Alexander Nadel. Solving huge instances with intel® sat solver: Supplementary material. https://technionmail-my.sharepoint.com/:f:/g/personal/alexandernad_technion_ac_il/EtihJbiS1XBJoi0DRTGbCnIBlfs1__hkPRiZ3uTpIR_x_g, 2023.

21 Alexander Nadel and Vadim Ryvchin. Efficient SAT solving under assumptions. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012 – 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 242–255. Springer, 2012. doi:10.1007/978-3-642-31612-8_19.

22 Collin Peterson. How to define and work with an array of bits in C?, November 2020. URL: https://stackoverflow.com/a/30590727.

23 Naveed A. Sherwani. *Algorithms for VLSI physical design automation.* Kluwer, 3 edition, November 1998.

24 Mate Soos. Allow memory to grow larger than 4gb per thread. https://github.com/msoos/cryptominisat/issues/389, May 2017.

25 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing – SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 – July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. doi:10.1007/978-3-642-02777-2_24.

26 Marc Thurley. sharpSAT – Counting models with advanced component caching and implicit BCP. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing – SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 424–429. Springer, 2006. doi:10.1007/11814948_38.

27 Takahisa Toda. Implementing efficient all solutions SAT solvers. *J. Exp. Algorithmics*, 21:1, 2016. doi:10.1145/2975585.

Learning Shorter Redundant Clauses in SDCL Using MaxSAT

Albert Oliveras
Technical University of Catalonia, Barcelona, Spain

Chunxiao Li
University of Waterloo, Canada

Darryl Wu
University of Waterloo, Canada

Jonathan Chung
University of Waterloo, Canada

Vijay Ganesh
University of Waterloo, Canada

Abstract

In this paper we present the design and implementation of a Satisfaction-Driven Clause Learning (SDCL) SAT solver, MapleSDCL, which uses a MaxSAT-based technique that enables it to learn shorter, and hence better, redundant clauses. We also perform a thorough empirical evaluation of our method and show that our SDCL solver solves Mutilated Chess Board (MCB) problems significantly faster than CDCL solvers, without requiring any alteration to the branching heuristic used by the underlying CDCL SAT solver.

2012 ACM Subject Classification Theory of computation

Keywords and phrases SAT, SDCL, MaxSAT

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.18

Funding *Albert Oliveras*: Supported by grant PID2021-122830OB-C43, funded by MCIN/AEI/10.13039/501100011033 and by "ERDF: A way of making Europe".

1 Introduction

Conflict-Driven Clause Learning (CDCL) SAT solvers are routinely used to solve large industrial problems obtained from variety of applications in software engineering [7], formal methods [8], security [11, 25] and AI [6], even though the underlying Boolean satisfiability (SAT) problem is well known to be NP-complete [9] and believed to be intractable in general. Despite this, solver research has made significant progress in improving CDCL solvers' components and heuristics [19].

It is well known that CDCL SAT solvers are polynomially equivalent to resolution [20, 1], and consequently it follows that classes of formulas, such as the pigeon hole principle (PHP), that are hard for resolution are also hard for CDCL SAT solvers. In order to address such limitations, researchers are actively designing and implementing solvers that correspond to stronger propositional proof systems.

One such class of solvers is called Satisfaction-Driven Clause Learning (SDCL) solvers [15, 14, 13], which are based on the propagation redundancy (PR) property [12, 14]. The SDCL paradigm extends CDCL in the following way: unlike CDCL solvers, SDCL solvers may learn clauses even when an assignment trail α is consistent. To be more precise, an SDCL solver first computes a new formula $P_\alpha(F)$, known as a pruning predicate. Then, it checks the satisfiability of $P_\alpha(F)$. If it satisfiable, it means $\neg\alpha$ is *redundant* with respect to the formula,

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 18; pp. 18:1–18:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and the solver can learn the clause $(\neg\alpha)$. Even though the intuition is clear and procedures for computing a possible $P_\alpha(F)$ are very well defined, it is still an extremely challenging task to automate SDCL.

There are two main problems in this setting: first, the satisfiability check for the formula $P_\alpha(F)$ is NP-complete and is hard to solve in general. It essentially requires the SDCL solver to call another SAT solver that we refer to as a sub-solver. Given that this sub-solver call can be expensive, one needs to be strategic about when to invoke it during the run of an SDCL solver. Second, the clauses learned by SDCL can be large, and we want to learn shorter clauses whenever possible.

To solve this second problem, we propose a novel MaxSAT encoding of the problem of "what is the smallest subset γ of trail α, such that $P_\gamma(F)$ is satisfiable", to get the shortest clause $(\neg\gamma)$ to learn. We also apply a resolution-based technique inspired by conflict analysis to further shorten the clause. We refer to the SDCL solver augmented with our MaxSAT and clause minimization technique as MapleSDCL. Our experimental evaluation shows that MapleSDCL performs well on mutilated chess board (MCB) and bipartite perfect matching problems, that are known to be hard for CDCL solvers.

1.1 Contributions

(I) First, we make a theoretical contribution by introducing a new type of pruning predicate and a proof that it allows one to detect blocked clauses. This extends the spectrum of pruning predicates with redundancy notions associated with them. However, we remark that this is not implemented in our system as we consider it to have little practical impact.

(II) Second, we prove that when an assignment has a satisfiable positive reduct, finding a small sub-assignment with the same property is an NP-hard problem. Such small assignments summarize the reasons for the redundancy and lead to learning smaller redundant clauses. In essence, we believe that this is the equivalent to conflict analysis in CDCL solvers.

(III) Third, we introduce a MaxSAT encoding of the above-stated problem. Experimental results show that calling a MaxSAT solver within the SDCL architecture is not as expensive as one might expect, and more importantly, significant improvements in the size of the learned redundant clauses are achievable in practice. These improvements are even larger after applying conflict analysis techniques to convert the clause into an asserting one.

(IV) Finally, we show that the resulting SDCL solver can solve mutilated chess board problems without the need to alter the decision heuristic used by the underlying CDCL SAT solver. This is a very important property of our approach: the chances of learning (good) redundant clauses depend much less on choosing exactly the right decision literals, thus overcoming a serious roadblock for SDCL solver design.

2 Preliminaries on CDCL SAT Solving

CNF formulas. Let $\mathcal{X}$ be a finite set of propositional *variables*. A *literal* is a propositional variable (x) or the negation of one $(\neg x)$. The *negation* of a literal l, denoted $\neg l$, is x if $l = \neg x$ and is $\neg x$ if $l = x$. A *clause* is a disjunction of distinct literals $l_1 \vee \ldots \vee l_n$ (interchangeably denoted with or without brackets). A *CNF formula* is a conjunction of distinct clauses $C_1 \wedge \ldots \wedge C_m$. When convenient, we consider a clause to be the set of its literals, and a CNF to be the set of its clauses. In the rest of the paper we assume that all formulas are CNF.

Satisfaction. An *assignment* is a set of non-contradictory literals. A *total* assignment contains, for each variable $x \in \mathcal{X}$, either x or $\neg x$. Otherwise, it is a *partial* assignment. We denote by $\neg\alpha$ the clause consisting of the negation of all literals in the assignment α. An assignment α satisfies a literal l if $l \in \alpha$, it satisfies a clause C if it satisfies at least one of the literals in C, and it satisfies a formula F if it satisfies all the clauses in F. We denote these as $\alpha \models l$, $\alpha \models C$, and $\alpha \models F$, respectively. A *model* for a formula is an assignment that satisfies it. A formula with at least one model is *satisfiable*; otherwise, it is *unsatisfiable*. Given a formula F, the *SAT* problem consists of determining whether F is satisfiable. An assignment α falsifies a literal l if $\neg l \in \alpha$, falsifies a clause if it falsifies all its literals, and falsifies a formula if it falsifies at least one of its clauses. The truth values of literals, clauses, and formulas are *undefined* for an assignment if they are neither falsified nor satisfied. Given a clause C and an assignment α, we denote by $touched_\alpha(C)$ the disjunction of all literals of C that are either satisfied or falsified by α, by $untouched_\alpha(C)$ the disjunction of all undefined literals, and by $satisfied_\alpha(C)$ the disjunction of all satisfied literals.

Unit propagation. Given a formula F and an assignment α, unit propagation extends α by repeatedly applying the following rule until reaching a fixed point: if there is a clause with all literals falsified by α except one literal l, which is undefined, add l to α. If, as a result, a clause is found that is falsified by α (called *conflict*), the procedure stops and reports that a conflict clause has been found.

Formula relations. Two formulas F and G are *equisatisfiable*, denoted $F \equiv_{SAT} G$, if F is satisfiable if and only if G is satisfiable, and they are *equivalent*, denoted $F \equiv G$, if they are satisfied by the same total assignments. We write $F \vdash_1 G$ (*F implies G by unit propagation*) if for every clause $C \in G$ of the form $l_1 \vee \ldots \vee l_n$, it holds that unit propagation applied to $F \wedge \neg l_1 \wedge \ldots \wedge \neg l_n$ results in a conflict. We say that G is a logical consequence of F (written $F \models G$) if all models of F are models of G.

CDCL. The Conflict-Driven Clause Learning (CDCL) algorithm is the most successful procedure to-date for determining whether certain types of industrial formulas are satisfiable [19]. Let F denote such a formula. The CDCL procedure starts with an empty assignment α, which is extended and reduced in a last-in first-out (LIFO) way, by the following three steps until the satisfiability of formula is determined (see Algorithm 1 removing lines 9-12):

1. Unit propagation is applied.
2. If a conflict is found, a *conflict analysis* procedure [26] derives a clause C (called a *lemma*) which is a logical consequence of F. If C is the empty clause, we can conclude that F is unsatisfiable. Otherwise, it is guaranteed that by removing enough literals from α, a new unit propagation is possible due to C. This process is called *backjump*. Additionally, lemma C is conjuncted (*learnt*) with F, and the procedure returns to step (i).
3. If no conflict is found in unit propagation, either α is a total assignment (and hence it satisfies the formula), or an undefined literal is chosen and added to α (the branching step). The choice of this literal, called a *decision literal*, is determined by sophisticated heuristics [4] that can have a huge impact on performance of the CDCL procedure.

MaxSAT. Given a formula F, the *MaxSAT* problem consists of finding the assignment that satisfies the maximum number of clauses of F. Sometimes the clauses in F are split into *hard* and *soft clauses*, and in this case, the *Partial MaxSAT* problem consists of finding the assignment that satisfies all hard clauses and the maximum number of soft clauses.

3 Propagation Redundancy and SDCL

Despite their success on a variety of real-world applications [23, 21, 5, 16], CDCL SAT solvers have well-known limitations. Proof complexity techniques have established the polynomial equivalence between CDCL and general resolution [20, 1], the proof system with the inference rule that allows one to derive $C \vee D$ given two clauses of the form $l \vee C$ and $\neg l \vee D$. An important consequence of this equivalence is that if an unsatisfiable formula does not have a polynomial size proof by resolution, no run of CDCL can determine the unsatisfiability of the formula in polynomial time.

3.1 Propagation Redundancy

This limitation has motivated the search for extensions of CDCL solvers that may allow the resultant method to simulate more powerful proof systems. One example is the extended resolution proof system [24]: by allowing the introduction of new variables to resolution, it can produce polynomial size proofs of the pigeon-hole principle [10], which requires exponential-size resolution proofs otherwise. However, adding new variables would exponentially increase the search space of the formula. A newer direction [12, 14] tries to avoid the addition of new variables, and is instead based on the well-known notion of redundancy:

▶ **Definition 1.** *A clause C is* **redundant** *with respect to a formula F if F and $F \wedge C$ are equisatisfiable.*

In order to provide a more useful characterization of redundancy, we need some definitions.

▶ **Definition 2.** *Given an assignment α and a clause C, we define $\mathbf{C}|_\alpha = \top$ if $\alpha \models C$; otherwise $\mathbf{C}|_\alpha$ is the clause consisting of all literals of C that are undefined in α. For a formula F, we define the formula $\mathbf{F}|_\alpha = \{C|_\alpha \mid C \in F \text{ and } \alpha \not\models C\}$.*

▶ **Theorem 3** ([12], Theorem 1)**.** *A non-empty clause C is redundant with respect to a formula F if and only if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \models F|_\omega$.*

From a practical point of view, this characterization does not help much, because even if we know ω (known as the *witness*) it is hard to check whether the property holds. This is why a more limited notion of redundancy has been defined [12]:

▶ **Definition 4.** *A clause C is* **propagation redundant (PR)** *with respect to a formula F if there exists an assignment ω such that $\omega \models C$ and $F \wedge \neg C \vdash_1 F|_\omega$*

Note that since $F \wedge \neg C \vdash_1 F|_\omega$ implies $F \wedge \neg C \models F|_\omega$, any PR clause is redundant. Hence, we can add PR clauses to our formula in order to make it easier to solve without affecting its satisfiability. If we force ω to assign all variables in C but no other variable, we can obtain weaker but simpler notions of redundancy: if we force ω to satisfy exactly one literal of C, we obtain *literal-propagation redundant (LPR)* clauses; if allow ω to satisfy more than one literal of C, we obtain *set-propagation redundant (SPR)* clauses. Obviously, any LPR clause is SPR, and any SPR clause is PR, but none of these three notions are equivalent as the following examples show.

▶ **Example 5** ([12])**.** Let $F = \{x \vee y,\ x \vee \neg y \vee z,\ \neg x \vee z,\ \neg x \vee u,\ x \vee \neg u\}$ and $C = x \vee u$. The witness $\omega = \{x, u\}$ satisfies C and, since $F|_\omega = \{z\}$, it holds that $F \wedge \neg C \vdash_1 F|_\omega$, that is, unit propagation on $F \wedge \neg x \wedge \neg u \wedge \neg z$ results in a conflict. Hence, C is SPR w.r.t. F.

However, it is not LPR. The reason is that there are only two possible witnesses that satisfy exactly one literal of C: $\omega_1 = \{x, \neg u\}$ and $\omega_2 = \{\neg x, u\}$. But we have that both $F|_{\omega_1}$ and $F|_{\omega_2}$ contain, among others, the empty clause. Hence, $F \wedge \neg C \vdash_1 F|_{\omega_1}$ and $F \wedge \neg C \vdash_1 F|_{\omega_2}$ require that unit propagation on $F \wedge \neg C$, that is, $F \wedge \neg x \wedge \neg u$, results in a conflict, which is not the case.

▶ **Example 6** ([12]). Let $F = \{x \vee y, \neg x \vee y, \neg x \vee z\}$ and $C = (x)$. If we consider the witness $\omega = \{x, z\}$, we have that $F|_\omega = \{y\}$. It is obvious that $\omega \models C$ and also $F \wedge \neg x \vdash_1 y$. Thus, C is PR w.r.t. F. However it is not SPR because the only possible witness would be $\omega_1 = \{x\}$, but $F|_{\omega_1} = \{y, z\}$ and it does not hold that $F \wedge \neg x \vdash_1 z$.

3.2 SDCL and Reducts

It was proved in [12] that the proof system that combines resolution with the addition of PR clauses admits polynomial-sized proofs for the pigeon hole principle. However, it is not a trivial task to add this capability to CDCL solvers. This question was addressed with the development of Satisfiability-Driven Clause Learning (SDCL) [15]. The key notion in this new solving paradigm is the one of *pruning predicate*:

▶ **Definition 7.** *Let F be a formula and α an assignment. A* **pruning predicate** *for F and α is a formula $P_\alpha(F)$ such that if it is satisfiable, then the clause $\neg\alpha$ is redundant w.r.t. F.*

SDCL extends CDCL in the following way (See also Algorithm 1). Before making a decision, a pruning predicate for the assignment α and formula F is constructed. If satisfiable, we can learn $\neg\alpha$ and use it for backjump and continuing the search, hence pruning away the search tree without needing to find a conflict. This leads to the simple code in Algorithm 1, where removing lines 9 to 12 results in the standard CDCL algorithm, and where we can assume, for simplicity, that $analyzeWitness()$ returns $\neg\alpha$. More sophisticated versions of $analyzeWitness$ are discussed in the next Section.

Algorithm 1 The SDCL algorithm. Note that removing lines 9–12 results in the CDCL algorithm.

```
 1  $\alpha := \emptyset$
 2  while true do
 3      $\alpha := unitPropagate(F, \alpha)$
 4      if conflict found then
 5          $C := analyzeConflict()$
 6          $F := F \wedge C$
 7          if $C$ is the empty clause then return UNSAT
 8          $\alpha := backjump(C, \alpha)$
 9      else if $P_\alpha(F)$ is satisfiable then
10          $C := analyzeWitness()$
11          $F := F \wedge C$
12          $\alpha := backjump(C, \alpha)$
13      else
14          if all variables are assigned then return SAT
15          $\alpha := \alpha \cup Decide()$
16      end
17  end
```

We can understand SDCL as a parameterized algorithm, since the use of different pruning predicates $P_\alpha(F)$ leads to distinct types of SDCL algorithms with possibly different underlying proof systems. In the following, we summarize the contributions of [15, 13] and explain the different pruning predicates and the corresponding proof systems that are known.

▶ **Definition 8.** *Given formula F and a (partial) assignment α, the* **positive reduct** *$p_\alpha(F)$ is the formula $\neg\alpha \wedge G$, where $G = \{touched_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.*

That is, we only consider clauses satisfied by α, and among them, only the literals that are assigned. In [15] it is proved that $p_\alpha(F)$ is a valid pruning predicate. Moreover, a precise characterization of the redundancy achieved by $p_\alpha(F)$ is given: $p_\alpha(F)$ is satisfiable if and only if $\neg\alpha$ is set-blocked in F.

▶ **Definition 9.** *A clause C is* **set-blocked** *in a formula F if there exists a subset $L \subseteq C$ such that, for every clause D containing the negation of some literal in C, the clause $(C \setminus L) \vee \neg L \vee D$ contains two complementary literals.*

The results in [15] imply that a proof system based on resolution and set-blocked clauses has polynomial size proofs for the pigeon hole principle. It is also known [12] that set-blocked clauses are a particular case of SPR clauses. If one wants to obtain the full power of SPR clauses, the following pruning predicate is needed:

▶ **Definition 10.** *Given formula F and a (partial) assignment α, the* **filtered positive reduct** *$f_\alpha(F)$ is the formula $\neg\alpha \wedge G$, where $G = \{touched_\alpha(D) \mid D \in F \text{ and } F \wedge \alpha \not\vdash_1 untouched_\alpha(D)\}$.*

Again, a precise characterization of the power of $f_\alpha(F)$ is known [13]: $f_\alpha(F)$ is satisfiable if and only if $\neg\alpha$ is SPR with respect to F. Despite being harder to compute than $p_\alpha(F)$, the fact that $f_\alpha(F)$ is a subset of the clauses in $p_\alpha(F)$ makes it easier to check for satisfiability. Finally, another pruning predicate is given in [13] that achieves the full power of PR clauses, but it is not considered to be practical. We close this sequence of pruning predicates and their corresponding redundancy characterization with a novel pruning predicate and its corresponding redundancy notion.

▶ **Definition 11.** *Given formula F and a (partial) assignment α, the* **purely positive reduct** *$pp_\alpha(F)$ is the formula $\neg\alpha \wedge G$, where $G = \{satisfied_\alpha(D) \mid D \in F \text{ and } \alpha \models D\}$.*

Since all clauses in $pp_\alpha(F)$ are subclauses of clauses in $p_\alpha(F)$, whenever $pp_\alpha(F)$ is satisfiable, $p_\alpha(F)$ is also satisfiable. This proves that $pp_\alpha(F)$ is a pruning predicate, but we can be more precise about the notion of redundancy it corresponds to.

▶ **Definition 12.** *We say that a literal $l \in C$* **blocks** *C in F if an only if for every clause D in F containing literal $\neg l$, resolution between C and D gives a tautology. A clause C is* **blocked** *in F if and only if there exists some literal $l \in C$ that blocks C in F.*

▶ **Theorem 13.** *Given a formula F and an assignment α, the formula $pp_\alpha(F)$ is satisfiable if and only if the clause $\neg\alpha$ is blocked in F.*

Proof.

Left to right. let β be a model of $pp_\alpha(F)$. Since $\beta \models \neg\alpha$, we can take any literal $\neg l$ in $\neg\alpha$ satisfied by β. We now prove that $\neg l$ blocks $\neg\alpha$ in F. Let us consider a clause of the form $l \vee C \in F$. Since $l \in \alpha$ we have that $\alpha \models l \vee C$, and hence there is a clause of the form $l \vee satisfied_\alpha(C)$ in $pp_\alpha(F)$. Since $\beta \models pp_\alpha(F)$ and $\beta \models \neg l$, necessarily $\beta \models satisfied_\alpha(C)$. This means that C contains a literal from α different from l, and hence if we apply resolution between the clause $\neg\alpha$ and $l \vee C$ we obtain a tautology.

Right to left. Assume w.l.o.g. that the clause $\neg\alpha$ is blocked w.r.t. $\neg l$ in F. We prove that $\hat{\alpha} := \alpha \setminus \{l\} \cup \{\neg l\}$ is a model of $pp_\alpha(F)$. It is obvious that $\hat{\alpha}$ satisfies the clause $\neg\alpha \in pp_\alpha(F)$. Any other clause $D \in pp_\alpha(F)$ is of the form $satisfied_\alpha(C)$ for some $C \in F$ such that $\alpha \models C$. There are now in principle two cases:

(i) if D is not the unit clause l, it necessarily contains a literal from α different from l, and hence $\hat{\alpha}$ satisfies it.

(ii) If D is the unit clause l, this means that clause $C \in F$ does not contain any literal from α except for l. Thus, applying resolution between $\neg\alpha$ and C cannot give a tautology, contradicting the fact that $\neg\alpha$ is blocked w.r.t $\neg l$ in F. Hence, this case cannot take place. ◀

We finish this section with one important remark about the computation of reducts in SDCL: we need to add all already computed redundant clauses in the reduct computation when trying to find additional ones. Let us show why not doing this is incorrect. Given the satisfiable formula $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_4)$, the SDCL solver might first build the assignment $\alpha = \{x_1, \neg x_2\}$. Its positive reduct is $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$, which is satisfiable, and hence we learn the redundant clause $\neg x_1 \vee x_2$. If the solver now builds the assignment $\{\neg x_1, x_2\}$, the positive reduct w.r.t F is $(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$, which is again satisfiable and allows us to learn the clause $x_1 \vee \neg x_2$. However, adding the two learned redundant clauses to F makes it unsatisfiable. The solution is to build the second positive reduct w.r.t. F conjuncted with the first learned redundant clause. The corresponding reduct is $(x_1 \vee \neg x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$, which is now unsatisfiable and hence does not allow us to learn the second redundant clause.

A natural question that arises now is whether we also need to add all clauses that were derived using CDCL-style conflict analysis in a reduct. The answer is that we do not need to do so. The reason is that, given two formulas $G_1 \equiv G_2$, it holds that C is redundant w.r.t. G_1 if and only if C is redundant w.r.t G_2. Now, if the current formula that the SDCL solver has in its database is $F \wedge L \wedge R$, where F is the original formula, L are the lemmas derived by CDCL-style conflict analysis and R are the learned redundant clauses, it holds that $F \wedge L \wedge R \equiv F \wedge R$. Therefore, it is sufficient to compute redundant clauses w.r.t. $F \wedge R$ only. Having said that, it is better to compute reduntant clauses w.r.t $F \wedge R \wedge U$, where U denotes CDCL-derived unit clauses, because it results in smaller reducts and faster sub-solver calls. Note that for correctness, clauses in R are never deleted. This design decision prevents us from using off-the-shelf proof checkers like dpr-trim[1]. However, as we mention at the end of Section 5, this checker can be easily adapted.

4 Minimizing SDCL Learned Clauses

In Algorithm 1, we considered the function $AnalyzeWitness$ to always return $\neg\alpha$, which was correct due to the results presented in Section 3. However, adding the negation of the whole assignment results in a very large clause, and it is not a surprise that this is far from being useful in practice. Already in [15] it was proven that one can learn a much shorter clause: the negation of all decisions in α. We provide a simple proof that we use to justify that learning other clauses is also correct:

▶ **Theorem 14.** *Let F be a formula and C a clause that is redundant with respect to F. Any clause D obtained via resolution steps from $F \wedge C$ is also redundant with respect to F.*

[1] `https://github.com/marijnheule/dpr-trim`

Proof. Let us assume that F is satisfiable and prove that $F \wedge D$ also is. Since C is redundant w.r.t. F we know that $F \wedge C$ is satisfiable. We know that resolution generates logical consequences, and hence any model of $F \wedge C$ is also a model of $F \wedge C \wedge D$, which proves that $F \wedge D$ is satisfiable. ◀

It is well known that, if α is an assignment, starting from $\neg\alpha$ one can apply a series of resolution steps in order to derive a clause that only consists of decisions. If $\neg\alpha$ is redundant, the theorem proves that the decision-only clause is also redundant. However, learning the negation of all decisions is not the ideal situation for at least two reasons. The first one is that, according to experience from CDCL SAT solving, forcing the solver to learn clauses that only contain decisions leads to very poor performance in practice. It is certainly true that these clauses are small, but that is probably their only good property. The second reason is that not all decisions in α need to be present in the redundant clause. Similarly to what happens in CDCL, where usually not all decisions are responsible for a conflict, here not all decisions are responsible for the pruning predicate to be satisfiable. In order to fix these two issues, we modify *AnalyzeWitness* so that it finds the smallest subset $\gamma \subseteq \alpha$ for which $P_\gamma(F)$ is satisfiable. This allows us to learn the hopefully much shorter clause $\neg\gamma$ and address one of the open problems mentioned in [14]: "checking if a subset of a conflict clause is propagation redundant with respect to the formula under consideration."

▶ **Example 15.** Consider $F = (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_5) \wedge (\neg x_1 \vee \neg x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (x_3 \vee x_6 \vee \neg x_5)$ and assignment, $\alpha = \{x_1, x_4, x_5, \neg x_2\}$, where x_5 is the only non-decision. The positive reduct $p_\alpha(F)$ is $(\neg x_1 \vee \neg x_4 \vee \neg x_5 \vee x_2) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_5) \wedge (\neg x_1 \vee \neg x_4 \vee x_5) \wedge (x_2 \vee x_4 \vee \neg x_5)$ and is satisfiable. Hence we could learn the redundant clause $\neg x_1 \vee \neg x_4 \vee x_2$ consisting of the negation of the decisions. However the subset $\gamma = \{x_1, \neg x_2\} \subseteq \alpha$ also has satisfiable positive reduct: $(\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$ and hence we could learn the shorter clause $\neg x_1 \vee x_2$.

4.1 Hardness of Minimization

Unfortunately, as we prove, the problem of finding such small γ is NP-hard. Let us first formalize it as a decision problem:

▶ **Definition 16.** *TRAIL-MINIMIZATION: given a formula F, an assignment α and an integer $k \geq 0$, we want to know whether there is a subset $\gamma \subseteq \alpha$ of size k such that $p_\gamma(F)$ is satisfiable.*

Note that in the rest of the paper we focus on the positive reduct of the formula, and hence our approach allows us to obtain (short) set-blocked clauses. Before introducing the NP-hard problem that we use to prove the NP-hardness of TRAIL-MINIMIZATION, we need one definition:

▶ **Definition 17.** *Given α and β two assignments over the same variables $\{x_1, \ldots, x_n\}$, we say that $\beta < \alpha$ if $\beta \neq \alpha$ and for each $\neg x_i \in \alpha$ we also have that $\neg x_i \in \beta$.*

In other words, $\beta < \alpha$ if, considering an assignment as a sequence of n bits, the sequence of bits of β is pointwise smaller than the one of α.

▶ **Definition 18.** *SMALLER-MODEL[17]: given a formula F and a total model α of F, we want to know whether there is a total model β of F such that $\beta < \alpha$.*

In [17] it is proved that SMALLER-MODEL is NP-hard. Hence, a polynomial reduction from SMALLER-MODEL to TRAIL-MINIMIZATION proves that the latter is also NP-hard.

▶ **Theorem 19.** *TRAIL-MINIMIZATION is NP-hard.*

Proof. Given (F, α) an instance of SMALLER-MODEL, we can partition $\alpha = \alpha^+ \cup \alpha^-$, where α^+ contains all positive literals in α, and α^- contains all negative literals. The reduction amounts to constructing an instance of TRAIL-MINIMIZATION as follows: the formula is $\hat{F} := F \cup \alpha^-$ (that is, we add to F all negative literals in α as unit clauses), the assignment is $\hat{\alpha} := \alpha$ and the integer $k := |\alpha|$. This can be computed in polynomial time and has polynomial size. Let us now check that (F, α) is a positive instance of SMALLER-MODEL if and only if $(\hat{F}, \hat{\alpha}, k)$ is a positive instance of TRAIL-MINIMIZATION.

Left to right: we know, by definition of SMALLER-MODEL, that there exists an assignment $\beta \models F$ such that $\beta < \alpha$. Since obviously $\hat{\alpha} \subseteq \hat{\alpha}$ and $|\hat{\alpha}| = k$, if we prove that $\beta \models p_{\hat{\alpha}}(\hat{F})$ we are done. Clause $\neg\alpha$ is satisfied by β because $\beta \neq \alpha$. Since $\hat{\alpha}$ is a total assignment, we have that $touched_{\hat{\alpha}}(C) = C$ for any clause $C \in \hat{F}$, hence any clause in $p_{\hat{\alpha}}(\hat{F})$ is either (i) a unit clause consisting of a literal in α^-, which is satisfied by β because $\beta < \alpha$ implies that $\beta \supseteq \alpha^-$ or (ii) a clause $C \in F$, which is of course satisfied by β since β is a model of F.

Right to left: the only subset of $\hat{\alpha}$ of size k is $\hat{\alpha}$ itself. Let us assume that $p_{\hat{\alpha}}(\hat{F})$ is satisfied by a model β. Since $\neg\alpha$ is a clause in $p_{\hat{\alpha}}(\hat{F})$, we know that $\beta \neq \alpha$. Also, since $p_{\hat{\alpha}}(\hat{F})$ contains all negative literals of α as unit clauses, we know that $\beta \supseteq \alpha^-$. Altogether, this proves that $\beta < \alpha$. The only missing piece is to prove that β satisfies F. This is not difficult to see: since α is a model of F, and $touched_{\alpha}(C) = C$ for any clause C, all clauses in F belong to $p_{\hat{\alpha}}(\hat{F})$ and β necessarily satisfies them. ◀

4.2 A MaxSAT Encoding for Trail-Minimization

Knowing that TRAIL-MINIMIZATION is a difficult optimization problem, and being somehow similar to SAT, it is very natural to try solving it using MaxSAT. Given a formula F, and an assignment α, we describe a partial MaxSAT formula $mp_{\alpha}(F)$ whose solutions correspond to a smallest $\gamma \subseteq \alpha$ such that $p_{\gamma}(F)$ is satisfiable.

Before formally defining $mp_{\alpha}(F)$, let us explain the intuition behind it. The main idea is that we have to determine which literals we can remove from α, giving a new assignment γ, such that $p_{\gamma}(F)$ is satisfiable. For each literal l in α, we add an additional variable r_l that indicates whether l is removed. Hence, a truth assignment over these variables induces an assignment $\gamma \subseteq \alpha$. The key point is to construct a formula such that when restricted with r_l's, it is essentially equivalent to $p_{\gamma}(F)$.

More formally, let us assume $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$. We introduce three sets of additional variables:

- $\{r_1, r_2, \dots, r_m\}$: indicate whether α_i is **r**emoved from α.
- $\{p_1, p_2, \dots, p_m\}$: replace "**p**ositive" occurrences of α_i in $p(F, \alpha)$.
- $\{n_1, n_2, \dots, n_m\}$: replace "**n**egative" occurrences of α_i in $p(F, \alpha)$.

Given a clause C, we denote by $\hat{C}$ the result of replacing in C, for $i = 1 \dots m$, every occurrence of literal α_i by p_i and every occurrence of literal $\neg\alpha_i$ by n_i.

Our Partial MaxSAT formula $mp_{\alpha}(F)$ contains the following hard formulas (that can be easily converted into clauses), that enforce the semantics of the r, p and n variables:

$$\begin{array}{rcll} r_i & \rightarrow & \neg p_i & \text{for all } i = 1 \dots m \\ r_i & \rightarrow & \neg n_i & \text{for all } i = 1 \dots m \\ \neg r_i & \rightarrow & p_i = \alpha_i & \text{for all } i = 1 \dots m \\ \neg r_i & \rightarrow & n_i = \neg\alpha_i & \text{for all } i = 1 \dots m \end{array}$$

Intuitively, if r_i is false, and hence we do not to remove α_i from the assignment, then p_i is equivalent to α_i and n_i is equivalent to $\neg\alpha_i$. Otherwise, if r_i is removed, we force p_i and n_i to be false.

The rest of $mp_\alpha(F)$ is constructed by iterating over all clauses of $p_\alpha(F)$. For each clause $C \in p_\alpha(F)$, we add a set of hard clauses to $mp_\alpha(F)$, constructed as follows. If C is the clause $\neg\alpha$, we add the hard clause $\widehat{\neg\alpha}$. Otherwise C is of the form $S \vee D$, where S is the non-empty set of literals satisfied by α and D contains the remaining literals, which are touched, but not satisfied by α. The clauses to be added are:

$$\{\hat{S} \vee \hat{D} \vee r_i \mid i = 1 \dots m \text{ and } \alpha_i \in S\}$$

The idea here is that if we remove all literals in S from α, then C would not be satisfied and hence it should not appear in the positive reduct. The addition of the r_i's in the clauses guarantee that, if all of them are removed, and hence all r_i's are set to true, these clauses are all satisfied by the r_i's and hence they do not constrain the formula at all. On the other hand, if some literal in S is not removed, then the corresponding r_i is false and we essentially have the clause $\hat{S} \vee \hat{D}$, that is what we wanted to impose.

We want to note that we can obtain a smaller formula by, instead of adding multiple clauses of the form of $\hat{S} \vee \hat{D} \vee r_i$, introducing one auxiliary variable a_C for each clause $C = S \vee D$ and adding the clauses:

$$\begin{array}{ll} \hat{S} \vee \hat{D} \vee a_C & \\ \neg r_i \rightarrow \neg a_C & \text{for all } i = 1 \dots m \text{ such that } \alpha_i \in S \end{array}$$

Apart from these hard clauses and the hard ones imposing the semantics of r, p and n, our formula $mp_\alpha(F)$ is completed with the set of soft unit clauses $\{r_i \mid i = 1 \dots m\}$, expressing that we want to remove as many literals as possible while still satisfying the rest of the formula, which are hard clauses.

▶ **Example 20.** Let us revisit Example 15, where $\alpha = \{x_1, x_4, x_5, \neg x_2\}$ had $p_\alpha(F)$ satisfiable, but there was a smaller subset $\gamma = \{x_1, \neg x_2\}$ for which $p_\gamma(F)$ was also satisfiable. We use this example to illustrate our encoding. Let us consider that the variables related with x_i are p_i, n_i, r_i for $i \in \{1, 2, 4, 5\}$. We only show the hard clauses in $mp_\alpha(F)$ that are constructed from $p_\alpha(F)$. The implications defining the semantics of p, n, r are ignored, as well as the soft clauses, since those should be easy to understand.

$\mathbf{p}_\alpha(\mathbf{F})$	$\mathbf{mp}_\alpha(\mathbf{F})$
$\neg x_1 \vee \neg x_4 \vee \neg x_5 \vee x_2$	$n_1 \vee n_4 \vee n_5 \vee n_2$
$x_1 \vee x_2 \vee x_4$	$p_1 \vee n_2 \vee p_4 \vee r_1$
	$p_1 \vee n_2 \vee p_4 \vee r_4$
$x_1 \vee \neg x_2 \vee x_5$	$p_1 \vee p_2 \vee p_5 \vee r_1$
	$p_1 \vee p_2 \vee p_5 \vee r_2$
	$p_1 \vee p_2 \vee p_5 \vee r_5$
$\neg x_1 \vee \neg x_4 \vee x_5$	$n_1 \vee n_4 \vee p_5 \vee r_5$
$x_2 \vee x_4 \vee \neg x_5$	$n_2 \vee p_4 \vee n_5 \vee r_4$

Note that any literal x_1 is replaced by p_1 since $x_1 \in \alpha$. On the other hand, any literal x_2 is replaced by n_2 because $\neg x_2 \in \alpha$. The interesting fact is that if we set r_4, r_5 to true and r_1, r_2 to false, which means that we are removing x_4 and x_5 from α (hence obtaining γ) and propagate the implications, the hard clauses in $mp_\alpha(F)$ become equivalent to $p_\gamma(F)$. For example, take the first clause $n_1 \vee n_4 \vee n_5 \vee n_2$. Setting r_4 and r_5 to true causes the implications to unit propagate $\neg n_4$ and $\neg n_5$. Hence the clause is equivalent to $n_1 \vee n_2$. However, setting r_1 to false makes $n_1 = \neg x_1$ and setting r_2 to false makes $n_2 = x_2$. All in all, the clause is equivalent to $\neg x_1 \vee x_2$, which is the first clause of $p_\gamma(F)$.

If we take clause $n_2 \vee p_4 \vee n_5 \vee r_4$ we can see that it is satisfied due to r_4 and thus is not constraining the other variables. This is as expected, because if we remove x_4 from α, it no longer satisfies the clause $x_2 \vee x_4 \vee \neg x_5$ and hence it should not appear in the reduct.

Finally, the last case is a clause like $p_1 \vee n_2 \vee p_4 \vee r_1$. In this case r_1 is false and hence the last literal in the clause disappears. Also, since r_4 is true, it makes p_4 false due to the implications, and r_1, r_2 being false unit propagates $p_1 = x_1$ and $n_2 = x_2$, hence making the clause equivalent to $x_1 \vee x_2$, which is precisely the clause that appears in $p_\gamma(F)$. All in all, if we set the r variables to the appropriate values we can obtain the positive reduct of any subset of α. Below, we formally prove that this encoding is indeed correct.

▶ **Theorem 21.** *Given a formula F and an assignment $\alpha = \{\alpha_1, \ldots, \alpha_m\}$, it holds that the smallest subset $\gamma \subseteq \alpha$ such that $p_\gamma(F)$ is satisfiable has size $m - k$ if and only if the optimal solution to $mp_\alpha(F)$ satisfies k soft clauses.*

Proof. We prove something slightly stronger: there exists $\gamma \subseteq \alpha$ of size $m - k$ such that $p_\gamma(F)$ is satisfiable if and only if there exists an assignment that satisfies all hard clauses in $mp_\alpha(F)$ and exactly k soft clauses.

Left to right. let us consider $\gamma \subseteq \alpha$ of size $m - k$ with $p_\gamma(F)$ satisfiable, and let δ be a model for it. We build an assignment the satisfies all hard clauses in $mp_\alpha(F)$ and exactly k soft clauses as follows. The first remark is that $mp_\alpha(F)$ only consists of the variables r_i, p_i, n_i and the ones appearing in α_i, for $i = 1 \ldots m$ and hence we have to build an assignment β over those. For $i = 1 \ldots m$ we add r_i to β if $\alpha_i \notin \gamma$, and add $\neg r_i$ otherwise. Since there are k literals α_i not belonging to γ, it is clear that β satisfies exactly k soft clauses. The assignment β is completed by making it coincide with δ on the variables of γ and take arbitrary values for the variables of $\alpha \setminus \gamma$. If we now unit propagate these values on the implications that define the semantics of r, p and n, we complete β to define values for all p_i and n_i.

Let us now see that β satisfies the hard clauses in $mp_\alpha(F)$. The implications defining the semantics of the variables are obviously satisfied. Clause $\widehat{\neg\alpha}$ is also satisfied: we know that this clause is of the form $n_1 \vee n_2 \vee \ldots \vee n_m$. Since $\delta \models \neg\gamma$, there is a literal $\alpha_k \in \gamma$ such that $\delta \models \neg\alpha_k$. By the definition of β, we know that $\neg r_k \in \beta$ and hence the formula $\neg r_k \;\; \rightarrow \;\; n_k = \neg\alpha_k$ propagates n_k to be true in β and hence satisfy $\widehat{\neg\alpha}$.

Let us now take another clause $C \in mp_\alpha(F)$, which is necessarily of the form $\hat{S} \vee \hat{D} \vee r_i$, with $S \vee D \in p_\alpha(F)$ and r_i be such that $\alpha_i \in S$. If $\beta \models r_i$ we are done. Otherwise, it is because $\alpha_i \in \gamma$. Hence, the clause $S \vee D$ is satisfied by γ due to literal $\alpha_i \in S$ and $p_\gamma(F)$ contains the clause $touched_\gamma(S \vee D)$. Thus, $\delta \models touched_\gamma(S \vee D)$. Let us consider that case where δ satisfies $\alpha_j \in touched_\gamma(S \vee D)$ (the other case is that is satisfies some $\neg\alpha_j$ and the proof is similar). Since $\alpha_j \in \gamma$, we have that $r_j \notin \beta$ and the formula $\neg r_j \;\; \rightarrow \;\; p_j = \alpha_j$ guarantees that $\beta \models p_j$. We only have to realize that p_j is a literal in $\hat{S} \vee \hat{D}$, to conclude that $\beta \models C$.

Right to left. let us consider an assignment β that satisfies all hard clauses in $mp_\alpha(F)$ and exactly k soft clauses. We build a subset $\gamma \subseteq \alpha$ of size $m - k$ such that $p_\gamma(F)$ is satisfiable. As expected, γ is constructed by removing from α all α_i such that $\beta \models r_i$. It is obvious that $|\gamma| = m - k$, because β satisfies exactly k unit clauses of the form r_i.

In order to prove that $p_\gamma(F)$ is satisfiable, let us build an assignment δ that coincides with β over all variables in γ and prove that it is a model. The first clause in $p_\gamma(F)$ to consider is $\neg\gamma$. Since $\beta \models n_1 \vee \ldots \vee n_m$ and it satisfies the clauses $r_i \;\rightarrow\; \neg n_i$ it necessarily satisfies some n_i such that r_i is false. Due to the clause $\neg r_i \rightarrow n_i = \neg\alpha_i$, it also satisfies $\neg\alpha_i$. Since r_i is false in β we have that $\alpha_i \in \gamma$ and hence, by the definition of δ, it satisfies $\neg\alpha_i$. This proves that $\delta \models \neg\gamma$.

Let us now take another clause in $p_\gamma(F)$, which is necessarily of the form $\mathit{touched}_\gamma(C)$ for some $C \in F$ such that $\gamma \models C$. Since α is a superset of γ, obviously $\alpha \models C$, and hence a clause of the form $\mathit{touched}_\alpha(C)$ belongs to $p_\alpha(F)$. This clause in $p_\alpha(F)$ is of the form $S \vee D$, with S containing all literals satisfied by α, and thus we have in $mp_\alpha(F)$ hard clauses of the form $\hat{S} \vee \hat{D} \vee r_i$ for every i with $\alpha_i \in S$. If $\gamma \models C$ it is because it satisfies some $\alpha_i \in C$ with r_i being false in β. Hence, the existence of the clause $\hat{S} \vee \hat{D} \vee r_i$ implies that $\beta \models \hat{S} \vee \hat{D}$. We know that $\hat{S} \vee \hat{D}$ is a disjunction of positive p's and n's literals. Let us assume that it satisfies some p_k (the case n_k is similar). Note that r_k has to be false because otherwise the implications force p_k to be false. Hence β satisfies some α_k such that r_k is false and hence $\alpha_k \in \gamma$, which means that δ also satisfies α_k because they coincide over γ. Since $\alpha_k \in \gamma$, it belongs to $\mathit{touched}_\gamma(C)$ which is the clause that we wanted δ to satisfy. ◀

4.3 Practical Remarks

The previous encoding would allow us to learn the redundant clause $C := \neg\gamma$. However, SDCL (see Algorithm 1) requires C to be asserting (i.e. containing exactly one literal of the last decision level, and hence allowing it to unit propagate after backjumping). In order to achieve this property, we first observe that, being the negation of a subset of the current assignment, clause $\neg\gamma$ is a conflict. Hence, we can apply standard CDCL conflict analysis to it and obtain a clause that is asserting. For those familiar with SMT, this is essentially what DPLL(T)-based SMT solvers do when they analyze theory conflicts. Thanks to Theorem 14, we can guarantee that the final clause we obtain in this process is redundant and hence can be safely added. Moreover, as can be seen in Section 5, the size of this clause tends to be even smaller than $\neg\gamma$. In addition, this method allows us to learn clauses that are stronger than set-blocked clauses.

▶ **Example 22.** Let us consider $F = (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_2) \wedge (x_0 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (x_3 \vee x_6 \vee \neg x_5)$. Assume the SAT solver builds the assignment, from left to right, $\{\mathbf{x_0}, x_1, \neg x_2, x_4, \mathbf{x_5}\}$ where literals in bold are decisions. If we pick $\gamma = \{x_0, x_1, \neg x_2\}$, we can see that its reduct $(\neg x_0 \vee \neg x_1 \vee x_2) \wedge (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_2) \wedge (x_0 \vee x_2)$ is satisfiable. This mean that we can learn $\neg x_0 \vee \neg x_1 \vee x_2$. Now, in two resolution steps with the reasons of x_1 and $\neg x_2$ which are $\neg x_0 \vee x_1$ and $\neg x_0 \vee \neg x_2$, respectively, we can derive the redundant clause $\neg x_0$. However, assignment x_0 does not have satisfiable positive reduct. In fact, clause $\neg x_0$ is not even SPR. It can be checked that it is indeed PR (a possible witness is $w = \{\neg x_0, x_2, x_3\}$). This shows that by combining the positive reduct with posterior resolution steps, we can obtain clauses with stronger redundancy properties than set-blocked clauses, which is the one obtained by using the positive reduct alone.

One final question that we want to address is whether, in an SDCL implementation, we should (i) first ask a SAT solver whether $p_\alpha(F)$ is satisfiable, and then, if this is the case, ask a MaxSAT to possibly find a smaller $\gamma \subseteq \alpha$ for which p_γ is also satisfiable, or (ii) directly ask a MaxSAT solver whether there exists a subset of $\gamma \subseteq \alpha$ for which p_γ is satisfiable. The following result sheds some light on this:

▶ **Proposition 23.** *Given a formula F and an assignment α, if $mp_\alpha(F)$ has some solution, then $p_\alpha(F)$ is satisfiable.*

Proof. By Theorem 21, if $mp_\alpha(F)$ has some solution satisfying k soft clauses, we can build an assignment $\gamma \subseteq \alpha$ for which $p_\gamma(F)$ is satisfiable, and let β be a model for it. It is now easy to prove that $\delta := \beta \cup \alpha \backslash \gamma$ is a model for $p_\alpha(F)$. The first clause in $p_\alpha(F)$ is $\neg\alpha$, of which the clause $\neg\gamma \in p_\gamma(F)$ is a subclause and hence $\beta \models \neg\alpha$. This implies that $\delta \models \neg\alpha$. Now, any other clause C in $p_\alpha(F)$ is of the form $touched_\alpha(D)$ for some $D \in F$ such that $\alpha \models D$. Expressing $touched_\alpha(D)$ as $touched_\gamma(D) \vee touched_{\alpha\backslash\gamma}(D)$ helps in our reasoning. If $\gamma \models D$ then $touched_\gamma(D) \in p_\gamma(F)$ and hence β satisfies it. In this case $\delta \models C$. Otherwise, $\gamma \not\models D$ but since $\alpha \models D$ it has to be that $\alpha\backslash\gamma \models D$. This implies that $\alpha\backslash\gamma \models touched_{\alpha\backslash\gamma}(D)$ and hence $\delta \models C$. ◀

This result shows that by directly calling the MaxSAT solver on $mp_\alpha(F)$, the solver cannot learn more redundant clauses than if we call the SAT solver on $p_\alpha(F)$. Hence, it makes sense to first call the SAT solver, which should be faster and then, only if $p_\alpha(F)$ has been found to be satisfiable, call the MaxSAT solver to possibly learn a shorter redundant clause. If we make an analogy with CDCL, checking $p_\alpha(F)$ for satisfiable would be the equivalent of unit propagation and solving the MaxSAT formula $mp_\alpha(F)$ the equivalent of conflict analysis.

5 Experimental Evaluation

5.1 Implementation

We implemented SDCL with the clause minimization techniques described in the previous section on top of the SAT solver MapleSAT [18]. In order to solve the MaxSAT queries, we have used EvalMaxSAT [2], an efficient solver that provides a very convenient C++ API.

The changes in Algorithm 1 are limited to $analyzeWitness$. Once we know that $p_\alpha(F)$ is satisfiable, we construct $mp_\alpha(F)$ and obtain the optimal solution with EvalMaxSAT. This induces a clause $\neg\gamma$, to which standard CDCL conflict analysis is applied in order to derive an asserting clause, which is learned and used to backjump.

This general idea is refined in different directions. First of all, we do not apply this procedure before every decision. Without redundant-clause minimization, this might be a bad decision design, since the length of the learned clause coincides with the decision level, and hence we should apply it as soon as possible. With clause minimization enabled, applying this procedure at high decision levels can still give short redundant clauses. Since, as a consequence of Proposition 23, we know that long assignments are more likely to produce redundant clauses, it makes sense to delay the check until the assignment is large enough. However, there is a certain trade-off because at high decision levels, $p_\alpha(F)$ and $mp_\alpha(F)$ are larger formulas and hence can be more difficult to solve. Our strategy relies on defining a decision level goal and trying to derive a redundant clause only when we are at this decision level. We compute the ratio of success (i.e. a redundant clause has been derived) of the procedure calls; if this ratio is lower than a certain amount (e.g. 15%), we increment the decision level goal; if it is higher, we decrement it. The rationale for this strategy is to achieve a predefined ratio of successful calls but not invoking the technique too often.

The second refinement is that our final asserting clause is not always shorter than the clause obtained by negating all decisions. In those rare situations, we learn the only-decisions clause. A final refinement consists of only learning clauses of size at most 3. In our SDCL implementation, we cannot delete redundant clauses we have learned in SDCL unless we also delete all CDCL learned clauses that have been derived using them. This is why we have to be very cautious and only learn high-quality redundant clauses.

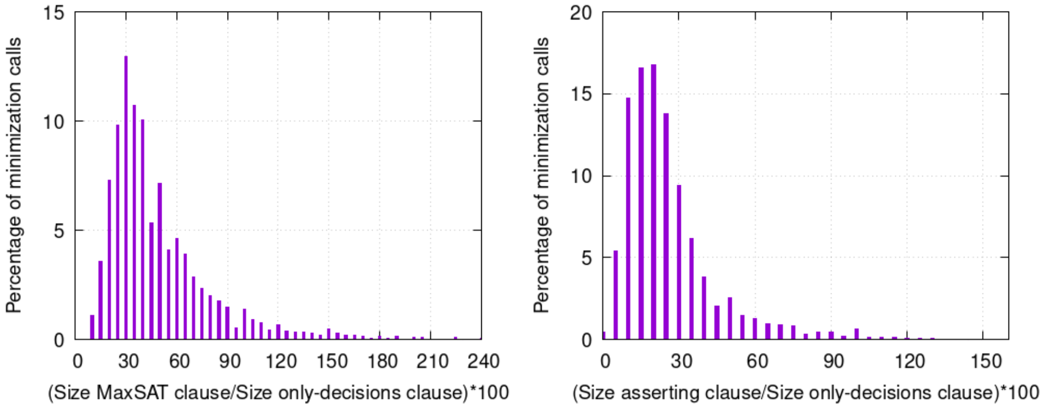


Figure 1 Distribution of the amount of minimization achieved in the clause returned by the MaxSAT solver (left) and in the final asserting clause (right).

One final remark is that, unlike previous SDCL implementations [15, 13], we have not modified the decision heuristics of the solver. We believe that, due to our conflict minimization techniques, picking the exact right variable at low decision levels is not so critical.

5.2 Experimental Results

We have evaluated our system on the benchmarks used in [13, 22]. In order to assess the impact of our Max-SAT based minimization technique, we have presented in Figure 1 results about one execution of our system on a mutilated chess board benchmark of size 20. Data for other benchmarks follow along the same lines. On the left-hand histogram, a bar over the x-point 30 with height 10 means that, in 10% of the calls to minimization, the percentage (Size MaxSAT clause / Size Only-decisions clause)*100 is between 30% and 35%. That is, the size of the MaxSAT clause was around one third the size of the only-decisions clause. The histogram on the right plots the same data, but comparing the final asserting clause with respect to the only-decisions clause. One can observe that the percentage of reduction is important and comes from the MaxSAT invocation as well as from the subsequent conflict analysis call that returns the final asserting clause.

We also studied the cost of calling the SAT solver for checking the satisfiability of $p_\alpha(F)$ and the MaxSAT solver for processing $mp_\alpha(F)$. Our experiments revealed that the cost of the SAT solver call never exceeds 2% of the total runtime, whereas the calls to MaxSAT are more expensive and they can account for almost 30% of the total runtime.

Finally, we present in Table 1 results on the performance of our system compared to others. We want to remark that no change to the decision heuristic of the baseline solver has been made. We chose Kissat [3] as a representative of a state-of-the-art CDCL SAT solver; SaDiCaL [15, 13] as the only other existing SDCL system; and our system MapleSDCL. For SaDiCaL, we used two versions, one using the positive reduct and one using the filtered positive reduct. Regarding MapleSDCL, we present three configurations: CDCL corresponds to the standard MapleSAT solver, implementing CDCL; SDCL represents a configuration using the positive reduct but no MaxSAT-based minimization, i.e., learning the only-decisions clause, and finally, SDCL-min uses the MaxSAT-based minimization presented in this paper. The table reports the number of seconds needed to solve each benchmark for each system. Due to the use of internal time limits in EvalMaxSAT, the exact behavior of SDCL-min is not reproducible. In order to have a higher confidence in its results we have run it 10

Table 1 Performance of different systems on mutilated chess board and bipartite perfect matching problems. Times are in seconds.

		SaDiCaL		MapleSDCL		
Benchmark	Kissat	Positive	Filtered	CDCL	SDCL	SDCL-min
mchess14	4.6	5682	3.6	11.7	7.3	2.7 (10)
mchess15	30.1	> 7200	13.8	54.3	24.7	5.5 (10)
mchess16	107	> 7200	19.5	439	191	9 (10)
mchess17	2293	> 7200	64.8	5038	517	25.8 (10)
mchess18	352	> 7200	71.8	> 7200	3803	52.8 (10)
mchess19	> 7200	> 7200	> 7200	> 7200	3578	128 (10)
mchess20	3720	> 7200	> 7200	> 7200	> 7200	369 (10)
mchess21	> 7200	> 7200	> 7200	> 7200	> 7200	977 (10)
mchess22	> 7200	> 7200	> 7200	> 7200	> 7200	4507 (7)
mchess23	> 7200	> 7200	> 7200	> 7200	> 7200	6041 (2)
randomG-Mix-17	> 7200	> 7200	> 7200	2837	1916	257 (10)
randomG-Mix-18	> 7200	> 7200	> 7200	> 7200	> 7200	1683 (10)
randomG-n17	> 7200	> 7200	> 7200	1266	688	157 (10)
randomG-n18	> 7200	> 7200	> 7200	> 7200	> 7200	2350 (10)

times on each benchmark. For this system, the number in parenthesis corresponds to the number of executions that solved that instance within the time limit of 7200 seconds, and the runtime is the average over those successful executions.

For pigeon-hole problems, we observed the same behavior reported in [13], dedicated decision heuristics are needed and they do not even work if the formula is scrambled. Tseitin formulas, and other benchmarks from the SAT competition used in [22] are out of reach of our system, probably to the fact that our current minimization procedure uses the positive reduct, and not the filtered one. All in all, we observed that our technique gives important benefit on mutilated chess board and bipartite perfect matching problems, outperforming all other competitors. We want to remark that the data showed in [22] indicate that their preprocessing-based technique for detecting PR clauses is able to achieve better performance. However, our goal was to show how far the SDCL framework can be improved, and we believe that results confirm that there is still a large space for improvement.

Finally, we would like to mention that MapleSDCL is able to produce proofs that are checkable with **dpr-trim**. However, this checker assumes that PR clauses are computed with respect to the current formula, including all learned lemmas. As already explained, we compute clauses that are PR with respect to $F \wedge R \wedge U$, where F is the initial formula, R contains all redundant clauses we have learned, and U is the set of all CDCL-like unit lemmas. This has forced us to add simple 6 lines of code to the checker that control which clauses have to be used when checking that the added PR clauses are correct.

6 Conclusions and Future Work

We have shown how redundant clauses learned within the SDCL approach can be shortened by encoding the problem as a partial MaxSAT formula. Via extensive empirical evaluation we show that our technique greatly improves the performance of SDCL over families of formulas for which it was theoretically known that SDCL had a competitive advantage with respect to CDCL. We outline several directions for future work. First of all, we could adapt the technique to also work for the filtered positive reduct. Secondly, there is a very interesting research opportunity in developing sophisticated adaptive strategies aimed at deciding as to when the SDCL solver should attempt to learn a redundant clause. Finally, parallelization of the MaxSAT calls would greatly improve the runtime of SDCL-based systems.

References

1 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011. doi:10.1613/jair.3152.

2 Florent Avellaneda. A short description of the solver EvalMaxSAT. In Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins, editors, *MaxSAT Evaluation 2020*, pages 8–9, 2020.

3 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

4 Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing – SAT 2015 – 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, 2015. doi:10.1007/978-3-319-24318-4_29.

5 Armin Biere and Daniel Kröning. Sat-based model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 277–303. Springer, 2018. doi:10.1007/978-3-319-10575-8_10.

6 Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1-2):281–300, 1997.

7 Cristian Cadar, Vijay Ganesh, Peter M Pawlowski, David L Dill, and Dawson R Engler. EXE: Automatically Generating Inputs of Death. *ACM Transactions on Information and System Security (TISSEC)*, 12(2):1–38, 2008.

8 Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model Checking.* MIT press, 2018.

9 Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971. doi:10.1145/800157.805047.

10 Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, 1976. doi:10.1145/1008335.1008338.

11 Julian Dolby, Mandana Vaziri, and Frank Tip. Finding Bugs Efficiently With a SAT Solver. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 195–204, 2007. doi:10.1145/1287624.1287653.

12 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In Leonardo de Moura, editor, *Automated Deduction – CADE 26 – 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2017. doi:10.1007/978-3-319-63046-5_9.

13 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Encoding redundancy for satisfaction-driven clause learning. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 41–58. Springer, 2019. doi:10.1007/978-3-030-17462-0_3.

14 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. *J. Autom. Reason.*, 64(3):533–554, 2020. doi:10.1007/s10817-019-09516-0.

15 Marijn J. H. Heule, Benjamin Kiesl, Martina Seidl, and Armin Biere. Pruning through satisfaction. In Ofer Strichman and Rachel Tzoref-Brill, editors, *Hardware and Software: Verification and Testing – 13th International Haifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15, 2017, Proceedings*, volume 10629 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2017. doi:10.1007/978-3-319-70389-3_12.

16 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016. doi:10.1007/978-3-319-40970-2_15.

17 Lefteris M. Kirousis and Phokion G. Kolaitis. The complexity of minimal satisfiability problems. *Inf. Comput.*, 187(1):20–39, 2003. doi:10.1016/S0890-5401(03)00037-3.

18 Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2016. doi:10.1007/978-3-319-40970-2_9.

19 João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2021. doi:10.3233/FAIA200987.

20 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. doi:10.1016/j.artint.2010.10.002.

21 Mukul R. Prasad, Armin Biere, and Aarti Gupta. A survey of recent advances in sat-based formal verification. *Int. J. Softw. Tools Technol. Transf.*, 7(2):156–173, 2005. doi:10.1007/s10009-004-0183-4.

22 Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant. Preprocessing of propagation redundant clauses. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning – 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 106–124. Springer, 2022. doi:10.1007/978-3-031-10769-6_8.

23 João P. Marques Silva and Karem A. Sakallah. Invited tutorial: Boolean satisfiability algorithms and applications in electronic design automation. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, page 3. Springer, 2000. doi:10.1007/10722167_3.

24 Grigori S Tseitin. On the complexity of derivation in propositional calculus. *Automation of reasoning: 2: Classical papers on computational logic 1967–1970*, pages 466–483, 1983.

25 Yichen Xie and Alexander Aiken. Saturn: A SAT-Based Tool for Bug Detection. In *Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005*, pages 139–143, 2005. doi:10.1007/11513988_13.

26 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified boolean satisfiability solver. In Lawrence T. Pileggi and Andreas Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 442–449. ACM / IEEE Computer Society, 2002. doi:10.1145/774572.774637.

UpMax: User Partitioning for MaxSAT

Pedro Orvalho
INESC-ID, Instituto Superior Técnico, University of Lisbon, Portugal

Vasco Manquinho
INESC-ID, Instituto Superior Técnico, University of Lisbon, Portugal

Ruben Martins
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

It has been shown that Maximum Satisfiability (MaxSAT) problem instances can be effectively solved by partitioning the set of soft clauses into several disjoint sets. The partitioning methods can be based on clause weights (e.g., stratification) or based on graph representations of the formula. Afterwards, a merge procedure is applied to guarantee that an optimal solution is found.

This paper proposes a new framework called UpMax that decouples the partitioning procedure from the MaxSAT solving algorithms. As a result, new partitioning procedures can be defined independently of the MaxSAT algorithm to be used. Moreover, this decoupling also allows users that build new MaxSAT formulas to propose partition schemes based on knowledge of the problem to be solved. We illustrate this approach using several problems and show that partitioning has a large impact on the performance of unsatisfiability-based MaxSAT algorithms.

2012 ACM Subject Classification Computing methodologies → Optimization algorithms

Keywords and phrases Maximum Satisfiability, Formula partitioning, Graph-based methods

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.19

Related Version *Full Version*: `https://arxiv.org/abs/2305.16191`

Supplementary Material *Software (Source code)*: `https://github.com/forge-lab/upmax`
archived at `swh:1:dir:8a7b30760263ec86f044920cbec1d9b8631eec2e`

Funding This work was partially supported by NSF award CCF-1762363 and an Amazon Research Award. This work was also supported by Portuguese national funds through FCT under projects UIDB/50021/2020, PTDC/CCI-COM/2156/2021, 2022.03537.PTDC and grant SFRH/BD/07724/-2020, and by project ANI 045917 funded by FEDER and FCT.

1 Introduction

In the last decade, Maximum Satisfiability (MaxSAT) algorithmic improvements have resulted in the successful usage of MaxSAT algorithms in several application domains such as fault localization [23], scheduling [14], planning [55], data analysis [9], among other [22, 51, 18]. These improvements resulted from new algorithm designs [39] based on iterative calls to a highly efficient Satisfiability (SAT) solver. However, MaxSAT algorithms also take advantage of other techniques, such as effective encodings of cardinality constraints [53] or the incremental usage of SAT solvers [34].

Another technique for MaxSAT solving is to use partitioning on the soft clauses. For instance, several solvers use partitioning of soft clauses according to their weight [4], which are particularly effective when the MaxSAT instance encodes a lexicographic optimization problem [32]. For the particular case of partial MaxSAT, other techniques have been proposed, such as using a graph representation of the formula [42]. However, despite its success for some classes of benchmarks, graph-based partitioning has not been widely used mainly because (1) the graph representation may become too large to build or to process,

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 19; pp. 19:1–19:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

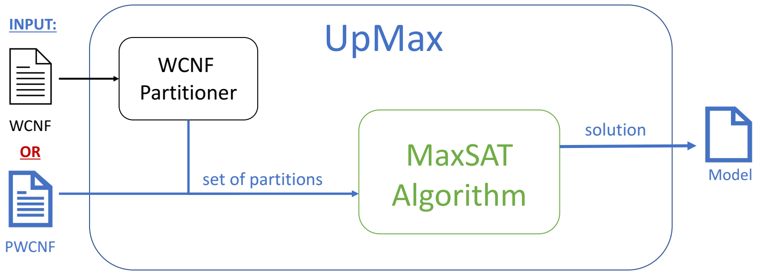


Figure 1 Overview of UpMax.

and (2) it is not decoupled with the base MaxSAT algorithm (i.e., changing the partition method implies altering the MaxSAT algorithm). Furthermore, in some cases, the partitions might not capture the problem structure that is helpful for MaxSAT solving. Since several MaxSAT algorithms rely on identifying unsatisfiable subformulas, each partition should be an approximation of an unsatisfiable subformula to be solved separately.

Until now, the partitioning of MaxSAT formulas is interconnected to the subsequent algorithm to be used. Therefore, it is not easy to define and test new partitioning methods with several MaxSAT algorithms developed by different people. The first contribution of this work is to propose a new format called `pwcnf` for defining MaxSAT formulas where clauses are split into partitions. Figure 1 illustrates the schematic view of the UpMax architecture based on the decoupling of the MaxSAT solving algorithm from the split of the clauses on the MaxSAT formula. Observe that any partitioning method (e.g., graph-based partitioning [42]) can be used to generate the instances in the new `pwcnf` format. Hence, this new format allows decoupling of the partitioning procedure from the MaxSAT algorithm, facilitating the appearance of new partition methods for MaxSAT formulas. Secondly, UpMax is not restricted to any partitioning scheme. UpMax also allows the MaxSAT user to propose how to partition MaxSAT formulas based on her domain knowledge of the problem to be solved. Note that this is not possible with current MaxSAT tools. Thirdly, with little effort, MaxSAT algorithms based on unsatisfiability approaches can be adapted to the new format. This is possible due to the newly proposed UpMax architecture. Hence, this paper presents the results of several algorithms using different partitioning schemes. Finally, we present several use cases where different user-defined partitioning schemes can be easily defined and tested. Experimental results show that user-based partitioning significantly impacts the performance of MaxSAT algorithms. Thus, UpMax decouples clause partitioning from MaxSAT solving, opening new research directions for partitioning, modeling, and algorithm development.

2 Background

A propositional formula in Conjunctive Normal Form (CNF) is defined as a conjunction of clauses where a clause is a disjunction of literals such that a literal is either a propositional variable v_i or its negation $\neg v_i$. Given a CNF formula ϕ, the Satisfiability (SAT) problem corresponds to decide if there is an assignment such that ϕ is satisfied or prove that no such assignment exists. The Maximum Satisfiability (MaxSAT) is an optimization version of the SAT problem. Given a CNF formula ϕ, the goal is to find an assignment that minimizes the number of unsatisfied clauses in ϕ. In partial MaxSAT, clauses in ϕ are split in hard ϕ_h and soft ϕ_s. Given a formula $\phi = (\phi_h, \phi_s)$, the goal is to find an assignment that satisfies all hard clauses in ϕ_h while minimizing the number of unsatisfied soft clauses in ϕ_s. The partial

Algorithm 1 Generic Partition-based MaxSAT Algorithm.

```
   Input: $\phi = (\phi_h, \phi_s)$
   Output: optimal assignment to $\phi$
1  $\gamma \leftarrow \langle \gamma_1, \ldots, \gamma_n \rangle \leftarrow$ partitionSoft($\phi_h, \phi_s$)     // initial partitions
2  if $|\gamma| = 1$ then
3      return MaxSAT ($\phi_h, \phi_s$)                                          // no partitions
4  foreach $\gamma_i \in \gamma$ do
5      $\nu \leftarrow$ MaxSAT($\phi_h, \gamma_i$)
6  while true do
7      $(\gamma_i, \gamma_j) \leftarrow$ selectPartitions($\gamma$)
8      $\gamma_k \leftarrow$ mergePartitions($\gamma_i, \gamma_j$)
9      $\gamma \leftarrow \gamma \setminus \{\gamma_i, \gamma_j\} \cup \{\gamma_k\}$                             // update partition set
10     $\nu \leftarrow$ MaxSAT($\phi_h, \gamma_k$)
11     if $|\gamma| = 1$ then
12         return $\nu$
```

MaxSAT problem can be further generalized to the weighted version, where each soft clause has an associated weight, and the optimization goal is to minimize the sum of the weights of the unsatisfied soft clauses. Finally, we assume that ϕ_h is satisfiable. Moreover, the set notation is also commonly used to manipulate formulas and clauses, i.e., a CNF formula can be seen as a set of clauses (its conjunction), and a clause as a set of literals (its disjunction).

2.1 Algorithms for MaxSAT

Currently, state of the art algorithms for MaxSAT are based on successive calls to a SAT solver [39]. One of the approaches is to perform a SAT-UNSAT linear search on the number of unsatisfied soft clauses. For that, the algorithm starts by adding a new relaxation variable r_i to each soft clause $s_i \in \phi_s$, where r_i represents the unsatisfiability of clause s_i. Next, it defines an initial upper bound μ on the number of unsatisfied soft clauses. At each SAT call, the constraint $\sum r_i \leq \mu - 1$ is added such that an assignment that improves on the previous one is found. Whenever the working formula becomes unsatisfiable, then the previous SAT call identified an optimal assignment. There are a plethora of these algorithms [26, 53, 44, 45].

On the other hand, UNSAT-SAT algorithms start with a lower bound λ on the number of unsatisfied soft clauses initialized at 0. The algorithm starts with an overconstrained working formula. At each iteration, the working formula is relaxed by adding additional relaxation variables allowing more soft clauses to be unsatisfied. Whenever the working formula becomes satisfiable, then an optimal solution is found. There are also many successful MaxSAT solvers that use an UNSAT-SAT approach [16, 4, 5, 31, 33, 40, 12]. Two key factors for the performance of UNSAT-SAT algorithms is the usage of SAT solvers to identify unsatisfiable subformulas, and the search process being incremental [15, 34]. Instead of dealing with the whole formula at once, some algorithms try to split the formula into partitions [4, 35, 42]. In particular, partitioning focuses on splitting the set of soft clauses into disjoint sets. The motivation is to quickly identify a minimal cost considering just a subset of soft clauses. Since the sets are disjoint, the sum of the minimal cost of all partitions defines a lower bound on the optimal solution. Moreover, a smaller instance should be able to be easier to solve. Hence, the convergence to the optimum is expected to be faster.

Algorithm 1 presents the pseudo-code for this generic approach. First, soft clauses in ϕ_s are split into n disjoint sets (line 1). If $n = 1$, then there is no partitioning and a MaxSAT solver is called on the whole formula. Otherwise, a MaxSAT solver solves each

Hard:	$h_1 : (v_1 \lor v_2)$	$h_2 : (\neg v_2 \lor v_3)$	$h_3 : (\neg v_1 \lor \neg v_3)$	$h_4 : (v_4 \lor v_5)$
	$h_5 : (\neg v_5 \lor v_6)$	$h_6 : (\neg v_4 \lor \neg v_6)$	$h_7 : (\neg v_3 \lor \neg v_6)$	
Soft:	$s_1 : (\neg v_1)$	$s_2 : (\neg v_3)$	$s_3 : (\neg v_4)$	$s_4 : (\neg v_6)$

Figure 2 Example of a MaxSAT formula.

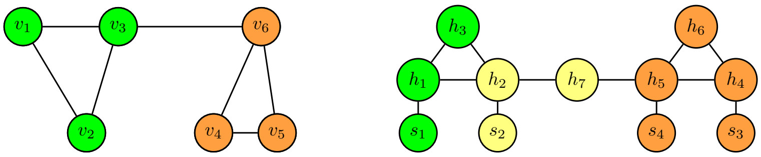


Figure 3 VIG graph (left) and RES graph (right) for MaxSAT formula in Figure 2.

partition γ_i independently (line 5). Next, two partitions are selected and merged (lines 7-8). The newly merged partition γ_k is then solved considering the information already obtained from solving γ_i and γ_j. This process is repeated until there is only one partition whose solution is an optimal assignment to the original MaxSAT instance (line 12). Observe that several MaxSAT algorithms can be used in this scheme including Fu-Malik [16], WPM3 [6], MSU3 [33], OLL [40] or a hitting set approach [12, 47], among others [39].

▶ **Example 1.** Consider the MaxSAT formula in Figure 2. Suppose the soft clauses are split into two disjoint sets $\gamma_1 = \{s_1, s_2\}$ and $\gamma_2 = \{s_3, s_4\}$. Next, a MaxSAT solver is applied to MaxSAT instances (ϕ_h, γ_1) and (ϕ_h, γ_2). Each of these instances has an optimal solution of 1. When merging both partitions, a final MaxSAT call is made on $(\phi_h, \gamma_1 \cup \gamma_2)$ with an initial lower bound of 2 (because γ_1 and γ_2 are disjoint). Since the lower bound is already equal to the optimum value, this last call is not likely to be computationally hard.

A related approach to partitioning is *Group MaxSAT* [7, 17]. Group MaxSAT is a variation of MaxSAT where soft clauses are grouped, and each group has a weight. The optimization goal in Group MaxSAT is to minimize the sum of the weights of the unsatisfied groups. A group is considered unsatisfied if at least one of its soft clauses is unsatisfied. Note that Group MaxSAT and MaxSAT are solving different optimization problems. The partitions in Algorithm 1 do not change the optimization goal of MaxSAT but instead are meant to guide the solver to find an optimal solution to the MaxSAT formula.

2.2 Partitioning MaxSAT Formulas

There are several graph representations for CNF formulas that have been proposed in order to analyze its structural properties [52, 50, 3]. For instance, it is well-known that industrial SAT instances can be represented in graphs with high modularity [3]. On the other hand, graphs that represent randomly generated instances are closer to an Erdös-Rényi model. Similar observations have also been made in MaxSAT instances [42]. Furthermore, based on these graph representations, one can partition the set of soft clauses in a MaxSAT instance by applying a community finding algorithm [10] that maximizes the modularity value.

One possible graph representation is the Variable Incidence Graph (VIG). Let $G = (V, E)$ denote a weighted undirected graph where V defines the graph vertices and E its edges. In the VIG representation, we have a vertex $v_i \in V$ for each variable v_i in the MaxSAT formula ϕ. Next, for each pair of variables v_i and v_j, if there is at least one clause in ϕ that contains both variables v_i and v_j (or its negated literals), then an edge (v_i, v_j) is added to the graph. For each clause $c \in \phi$ with n literals, then $1/\binom{n}{2}$ is added to the weight of every pair of variables that occurs in clause c. Figure 3 illustrates the VIG representation for the MaxSAT formula in Figure 2. Note that edge weights are not represented to simplify the figure. Next, if we apply a community finding algorithm that maximizes the modularity, two communities are identified (vertices with different colors). Hence, soft clauses with variables in $\{v_1, v_2, v_3\}$ would define a partition, while soft clauses with variables in $\{v_4, v_5, v_6\}$ define the other partition. Therefore, we would have $\gamma_1 = \{s_1, s_2\}$ and $\gamma_2 = \{s_3, s_4\}$ as the two partitions of soft clauses. Observe that $\phi_h \cup \gamma_1$ is an unsatisfiable subformula of ϕ, as well as $\phi_h \cup \gamma_2$. In the Clause-Variable Incidence Graph (CVIG) representation, there is a node for each variable and another node for each clause. Moreover, if a variable v_i (or its negation $\neg v_i$) occurs in a clause c_j, then there is an edge (v_i, c_j) in the graph.

On the other hand, in Resolution-based Graphs (RES) only clauses are represented as vertices. Hence, for each clause c_j there is a node in the graph. Let c^r_{jk} be the clause that results from applying the resolution operation between clauses c_j and c_k. If c^r_{jk} is not a tautology, then an edge (c_j, c_k) is added to the graph with weight $1/|c^r_{jk}|$ where $|c^r_{jk}|$ denotes the size of the resolvent clause. Note that if the resolution operation results in a trivial resolvent, then no edge is added. The right graph in Figure 3 shows the RES graph representation of the MaxSAT formula in Figure 2. Colors illustrate the three communities of soft clauses found in this formula, i.e., $\gamma_1 = \{s_1\}$, $\gamma_2 = \{s_2\}$ and $\gamma_3 = \{s_3, s_4\}$.

3 User-Based Partitioning

All the partitioning methods described in Section 2.2 are automatic and attempt to recover some partition scheme from the structure of the MaxSAT formula. However, when encoding a problem into MaxSAT, it is often the case that the user has enough domain knowledge to provide a potential partition scheme. Unfortunately, the current MaxSAT format does not support this extra information. Thus, we propose a new generic format for MaxSAT, `pwcnf`, where the partition scheme can be defined, and solvers can take advantage of it. The `pwcnf` format starts with a header:

```
p pwcnf n_vars n_clauses topw n_part
```

and each line in the body is of the form: `[part] [weight] [literals*] 0`.

In the header, `n_vars` and `n_clauses` are the numbers of variables and clauses of the formula. `topw` is the weight assigned to the hard clauses and `n_part` is the number of partitions. Each partition label (`[part]`) must be a positive integer from 1 to `n_part`.

▶ **Example 2.** Considering the RES graph presented in Figure 3, clearly it has 3 partitions accordingly to the coloring scheme: green (label 1), yellow (label 2) and orange (label 3). The `pwcnf` for this graph is the following:

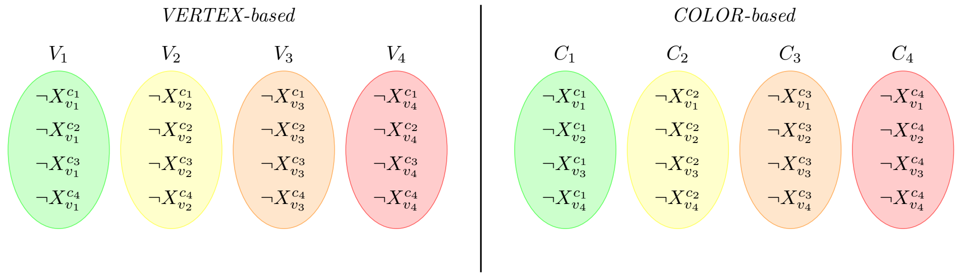
```
p pwcnf 6 11 7 3      1 7 -1 -3 0    3 7 -4 -6 0    2 1 -3 0
1 7 1 2 0             3 7 4 5 0      2 7 -3 -6 0    3 1 -4 0
2 7 -2 3 0            3 7 -5 6 0     1 1 -1 0       3 1 -6 0
```

Alloy [21] is a declarative modeling language based on a first-order relational logic that has been applied to different software engineering problems [25, 30, 24, 49]. Recently, Alloy^{Max} has been proposed that extends Alloy with the ability to find optimal solutions [54]. Alloy^{Max} uses a MaxSAT solver as its optimization engine. So for that, Alloy^{Max} encodes in a MaxSAT formula the high-level Alloy model. Moreover, Alloy^{Max} uses domain knowledge to partition the soft clauses in the generated MaxSAT formula. In particular, one partition of soft clauses is created for each optimization operator used in the Alloy specification. Alloy^{Max} was the first successful usage of UpMax, which shows that the creation of the new file format makes it easier for other researchers to integrate partitioning in their applications.

3.1 Use Case: Minimum Sum Coloring

To illustrate how a user can take advantage of the new proposed format, we analyze the Minimum Sum Coloring (MSC) problem from graph theory. MSC is the problem of finding a proper coloring while minimizing the sum of the colors assigned to the vertices. In this problem, the following conditions must be met: (1) each vertex should be assigned a color; (2) each vertex is assigned at most one color; and (3) two adjacent vertices cannot be assigned the same color. The optimization goal is to minimize the number of different colors in the graph. Let V denote the set of vertices in the graph. Let C denote the set of possible colors. Let X_c^v be the Boolean that is assigned to 1 if color c is assigned to vertex v. The goal is to maximize $\neg X_c^v$, and each soft clause is assigned the weight of c. The user can potentially group these soft clauses in two distinct ways: (1) variables that share the same color are grouped, or (2) variables that share the vertex number are grouped. For more details on the MaxSAT encoding, the reader is referred to the extended version of this paper [43].

▶ **Example 3.** Assume a user wants to minimize the number of different colors needed to color a given graph G such that two adjacent vertices cannot share the same color. G has 4 vertices, $v_1, \ldots, v_4$, and the following set of edges $G_E = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_3, v_4)\}$. Furthermore, there are 4 different colors available $c_1, \ldots, c_4$. When encoding the problem into `pwcnf` the user could provide either the following VERTEX-based or COLOR-based partition scheme:



3.2 Use Case: Seating Assignment Problem

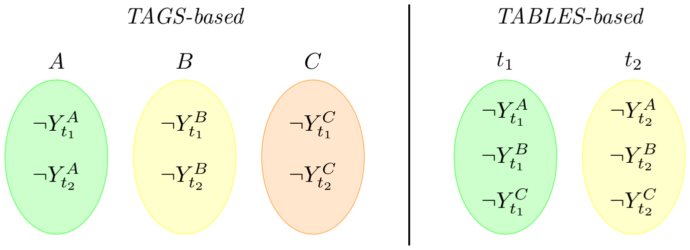
Another example of how a user can take advantage of the `pwcnf` format is the seating assignment problem. We encode this problem into MaxSAT and show different partition schemes that can be provided by the user. Consider a seating assignment problem where the goal is to seat persons at tables such that the following properties are met: (1) Each table has a minimum and a maximum number of persons; (2) Each person is seated at exactly one table; and (3) Each person has some tags that represent their interests. The optimization goal is to minimize the number of different tags between all persons seated at the same table.

More formally, consider a seating problem with p persons and t tables. Assume that the set of tags each person may have is defined by G and the set of tables is defined by T. Consider the Boolean variables Y_t^g that are assigned to 1 if there is *at least one person* p with a tag g that is seated at table t. The goal of this optimization problem is to minimize $\sum_{t \in T, g \in G} Y_t^g$ subject to the constraints of the problem. When encoding the problem into MaxSAT, the $\neg Y_t^g$ literals will correspond to unit soft clauses. The user can potentially group these soft clauses in two distinct ways: (1) variables that share the same tag are grouped, or (2) variables that share the same table are grouped. We call the former partition scheme *TAGS-based* and the latter *TABLES-based*. More details can be found in the extended version of this paper [43].

▶ **Example 4.** Consider that a user wants to seat 5 persons, $p_1, \ldots, p_5$, in two tables t_1, t_2. Each table must have at least 2 persons and at most 3 persons. Each person has a set of interests described by their tags as follows:

$$p_1 = \{A, B\}, p_2 = \{C\}, p_3 = \{B\}, p_4 = \{C, A\}, p_5 = \{A\}$$

When encoding the problem into `pwcnf` the user could provide either the following TAGS-based or TABLES-based partition scheme:



4 Experimental Results

UpMax is built on top of the open-source Open-WBO MaxSAT solver [36]. UpMax supports the new format `pwcnf` for user-based partitioning presented in Section 3. Alternatively, it can also take as input a `wcnf` formula and output a `pwcnf` formula using an automatic partitioning strategy based on VIG [37], CVIG [37], RES [42], or randomly splitting the formula into k partitions. UpMax can also be extended to support additional partitioning strategies that users may want to implement to evaluate their impact on the performance of MaxSAT algorithms. Furthermore, we are currently merging the partitions based on their size. We sort the partitions in increasing order, and we start by giving the smaller partition to the solver. However, other merging methods can be easily implemented and evaluated.

UpMax currently supports three UNSAT-based algorithms (WBO [31], OLL [40], and MSU3 [34]) for both unweighted and weighted problems that take advantage of the partitions using the basic algorithm described in Algorithm 1. WBO uses only at-most-one cardinality constraints when relaxing the formula at each iteration. In contrast, MSU3 uses a single cardinality constraint, and OLL uses multiple cardinality constraints. Furthermore, we have also extended RC2 [20] and Hitman [38], available in PySAT [19], to take advantage of user-based or graph-based partitions through our `pwcnf` formulae using Algorithm 1. RC2 is an improved version of the OLL algorithm [40, 41]. Hitman is a SAT-based implementation of an implicit minimal hitting set enumerator [38] and can be used as the basic flow of a MaxHS-like algorithm [11] for MaxSAT. UpMax is publicly available at GitHub [1].

To show the impact of partitioning on the performance of unsatisfiability-based algorithms, we randomly generate 1,000 instances for both the seating assignment and the minimum sum coloring problem by varying the different parameters of each problem. We considered the automatic partitioning strategies available in UpMax (VIG, CVIG, RES, random; for the random partitioning strategy, we fixed $k = 16$), the user partitions (UP) described in Sections 3.1 and 3.2, and no partitions. All of the experiments were run on StarExec [48] with a timeout of 1800 seconds and a memory limit of 32 GB for each instance.

4.1 Minimum Sum Coloring Problem

Table 1 shows the number of instances solved for the Minimum Sum Coloring problem for each partitioning strategy and algorithm. Entries highlighted in bold correspond to the highest number of instances solved for each algorithm. The diversity of algorithms and

Table 1 Number of solved instances for the Minimum Sum Coloring (MSC) problem.

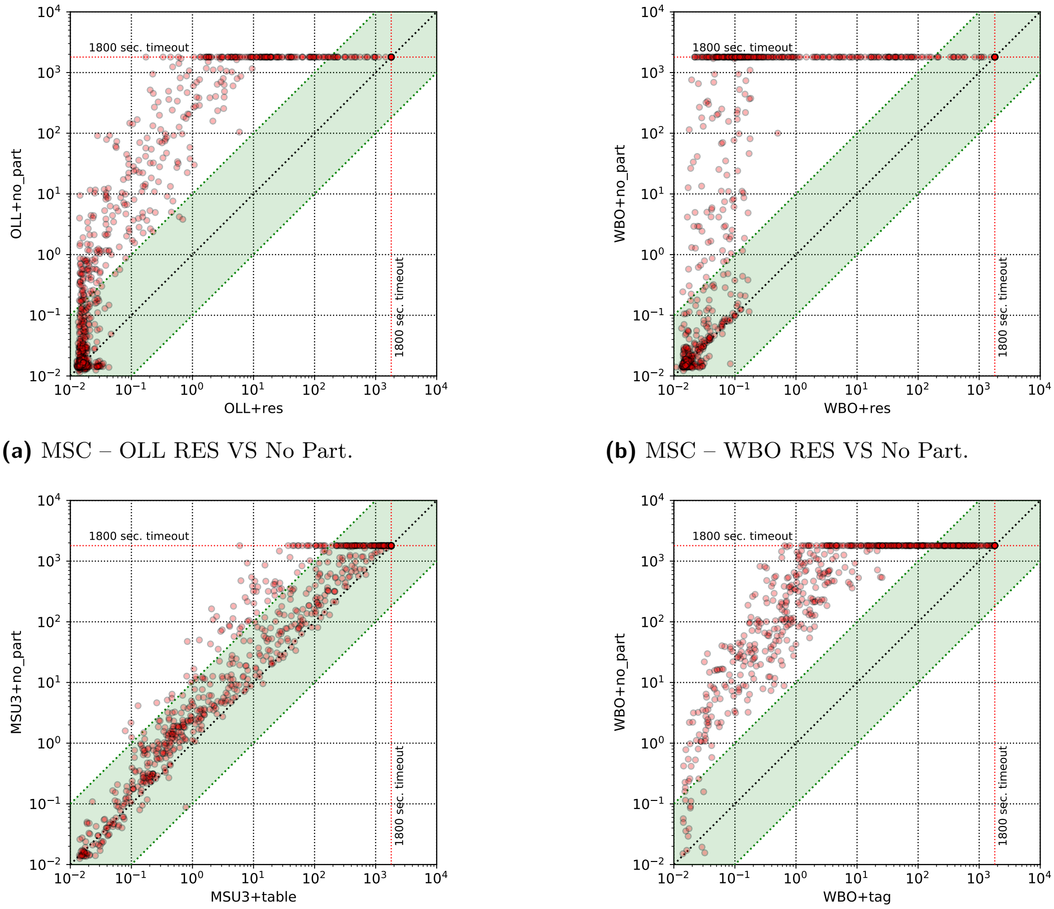
		User Part.		Graph Part.			
Solver	**No Part.**	**Vertex**	**Color**	**VIG**	**CVIG**	**RES**	**Random**
MSU3	245	758	770	774	770	775	**776**
OLL	796	863	594	945	944	**947**	756
WBO	483	622	314	745	750	**755**	493
Hitman	610	613	471	605	**614**	609	592
RC2	796	866	528	943	939	**944**	687

Table 2 Number of solved instances for the Seating Assignment problem.

		User Part.		Graph Part.			
Solver	**No Part.**	**Table**	**Tag**	**VIG**	**CVIG**	**RES**	**Random**
MSU3	558	**671**	639	659	641	640	565
OLL	526	**634**	624	627	599	608	528
WBO	306	400	**536**	400	385	386	360
Hitman	420	403	**510**	406	425	420	440
RC2	530	620	**624**	618	600	597	541

partitions used allows us to make some interesting observations regarding the impact of partitioning on the performance of MaxSAT algorithms. First, we can see that partitioning MaxSAT algorithms can often significantly outperform their non-partitioning counterparts. For instance, with partitioning the WBO algorithm can solve 272 more instances than without partitioning. Secondly, most partition schemes result in performance improvements, even if the partition is done randomly. In this problem, random partitions had a benefit for the MSU3 algorithm. This occurs since, until the last partition is added, this algorithm deals with a subset of soft clauses, resulting in finding smaller unsatisfiable cores. Another observation is that the user-based partitions were not as good as the graph-based partitions. This may be partially explained by the fact that the user-based partitions do not consider the weight of the soft clauses which is important for weighted MaxSAT algorithms. Finally, we can also observe that different partition strategies have different performance impacts on different algorithms. This suggests that new algorithms could leverage the partition information better than our approach presented in Algorithm 1.

Figures 4a and 4b show two scatter plots comparing two MaxSAT algorithms, OLL and WBO, on the set of instances for the Minimum Sum Coloring problem. Each point in Figure 4a represents an instance where the x-value (resp. y-value) is the CPU time spent to solve the instance using the OLL algorithm with the RES partitioning scheme (resp. OLL without any partitioning scheme). If a point is above the diagonal, then it means that the algorithm with partitioning outperformed the algorithm without partitioning. The OLL algorithm was the one with the best performance in the Minimum Sum Coloring (MSC) set of instances (see Table 1). For many instances, we can observe a 10× speedup for OLL-RES when compared to OLL-NoPart. Secondly, Figure 4b compares the WBO algorithm with no partitioning and WBO using the RES partitioning scheme. WBO has the most significant gap between using partitions (755 instances solved) and not using partitions (483 instances solved). Figure 4b also shows that many instances that could not be solved without partitions can now be solved in a few seconds. Both plots support that OLL and WBO greatly improve their performance on this set of benchmarks when using partitioning. Cactus plots of our experiments can be found in the extended version of this paper [43].



(a) MSC – OLL RES VS No Part.

(b) MSC – WBO RES VS No Part.

(c) SA – MSU3 Table VS No Part.

(d) SA – WBO Tag VS No Part.

Figure 4 Scatter plots comparing different MaxSAT algorithms and respective partitioning schemes for the problems of Minimum Sum Coloring (MSC) and Seating Assignment (SA).

4.2 Seating Assignment Problem

Table 2 shows the number of instances solved for the Seating Assignment problem for each partitioning strategy and algorithm. Similar to the previous use case, we can observe that partitioning can significantly impact the number of solved instances for all evaluated MaxSAT algorithms. For instance, WBO algorithm with partitioning can solve more 220 instances than without partitioning. Furthermore, almost all the presented partition schemes, except random partitioning, result in performance improvements when compared to no partitioning scheme. Moreover, different partition strategies have different performance impacts on different algorithms. In this problem, user-based partitions achieved the best results. However, in some algorithms (e.g., MSU3, OLL), tabled-based partitioning is the best approach, while tag-based partitioning is better for the other algorithms.

Figures 4c and 4d show two scatter plots comparing two MaxSAT algorithms, MSU3 and WBO, on the set of instances for the Seating Assignment problem. Figure 4c shows the effectiveness of the table-based partitioning scheme against not using partitions in the MSU3 algorithm. We can observe that partitioning leads to faster runtimes, with most points being above the diagonal. Secondly, Figure 4d compares the WBO algorithm with the tag-based and without any partitioning scheme. This algorithm has the biggest gap between using partitions (536 instances solved) and not using partitions (306 instances solved). Figure 4d supports that using partitioning with the WBO algorithm on this set of benchmarks greatly improves its performance with speedups of more than 10× for most of the instances.

4.3 State-of-the-art MaxSAT Solvers

Using the `wcnf` formulae (No Part.) of both benchmark sets, we compared the performance of UpMax with some of the best solvers [1] in the MaxSAT Evaluation 2022 [2], such as MaxHS [13], UWrMaxSat-SCIP [46], CASHWMaxSAT-CorePlus [27], EvalMaxSAT [8], and MaxCDCL [29]. MaxHS is a MaxSAT solver based on an implicit hitting set approach. UWrMaxSat is an unsatisfiability-based solver using the OLL algorithm. These solvers can be seen as better versions than the RC2 and Hitman algorithms available in PySAT. CASHWMaxSAT is developed from UWrMaxSat, EvalMaxSAT is based on the OLL algorithm, and MaxCDCL is an extension for MaxSAT of the CDCL algorithm [28], which combines Branch and Bound and clause learning.

Regarding the minimum sum coloring problem, MaxHS solved 873 instances, EvalMaxSAT solved 729, CASHWMaxSAT solved 993 (708 without SCIP), UWrMaxSat solved 994 (728 without SCIP), and MaxCDCL solved 995 instances. Note that solvers using Branch and Bound excel on these instances, and the performance of CASHWMaxSAT and UWrMaxSat deteriorates when SCIP is not used. Moreover, these results also show that partitioning improves less effective MaxSAT algorithms to become competitive with some solvers (e.g. UWrMaxSat), and outperform other solvers, e.g., MaxHS and EvalMaxSAT. Secondly, regarding the seating assignment problem, UWrMaxSat solved 580 instances, CASHWMaxSAT solved 585, MaxCDCL solved 593, MaxHS solved 643, and EvalMaxSAT solved 653 instances. When compared with our best results, note that table-based partitioning with the MSU3 algorithm can outperform all these solvers. Moreover, since partitioning can improve the performance of multiple MaxSAT algorithms and it is beneficial for implicit hitting set approaches like Hitman, it has the potential to further improve the performance of MaxHS.

Even though partitioning is not expected to improve the performance of MaxSAT solvers on all problem domains, there are many domains similar to the seating assignment and minimum sum coloring [2] for which partitioning can provide a significant performance boost in MaxSAT solving. Finally, we note this work opens new lines of research based on decoupling of MaxSAT solvers from the procedure that defines the partitions of MaxSAT formulae.

5 Conclusions

In this paper, we propose UpMax, a new framework that decouples the partition generation from the MaxSAT solving. UpMax allows the user to specify how to partition MaxSAT formulas with the proposed `pwcnf` format. With this format, the partitioning of MaxSAT instances can be done a priori to MaxSAT solving. Experimental results with two use cases with multiple algorithms show that partitioning can improve the performance of MaxSAT algorithms and allow them to solve more instances. UpMax provides an extendable framework that can benefit (1) researchers on partitioning strategies, (2) solver developers with new MaxSAT algorithms that can leverage partition information, and (3) users that can benefit from additional information when modeling problems to MaxSAT.

[1] We did not modify any of these solvers since each solver has a large codebase with many optimizations, but these solvers could also use a partitioning approach.

[2] Check Alloy$^{\text{Max}}$ [54] paper for UpMax's results on other application domains.

References

1 `https://github.com/forge-lab/upmax`. Last accessed on 28th February 2023.

2 `https://maxsat-evaluations.github.io/2022/`. Last accessed on 28th February 2023.

3 C. Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The Community Structure of SAT Formulas. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 7317 of *LNCS*, pages 410–423. Springer, 2012.

4 Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving SAT-Based Weighted MaxSAT Solvers. In *Principles and Practice of Constraint Programming*, volume 7514 of *LNCS*, pages 86–101. Springer, 2012.

5 Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving WPM2 for (Weighted) Partial MaxSAT. In *Principles and Practice of Constraint Programming*, volume 8124 of *LNCS*, pages 117–132. Springer, 2013.

6 Carlos Ansótegui and Joel Gabàs. WPM3: an (in)complete algorithm for weighted partial maxsat. *Artificial Intelligence*, 250:37–57, 2017.

7 Josep Argelich and Felip Manyà. Exact max-sat solvers for over-constrained problems. *J. Heuristics*, 12(4-5):375–392, 2006.

8 Florent Avellaneda. A short description of the solver evalmaxsat. *MaxSAT Evaluation*, 8, 2020.

9 Jeremias Berg, Antti Hyttinen, and Matti Järvisalo. Applications of maxsat in data analysis. In Daniel Le Berre and Matti Järvisalo, editors, *International Workshop Pragmatics of SAT*, volume 59 of *EPiC Series in Computing*, pages 50–64. EasyChair, 2018.

10 V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics*, 2008(10):P10008, 2008.

11 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming*, volume 6876 of *LNCS*, pages 225–239. Springer, 2011. `doi:10.1007/978-3-642-23786-7_19`.

12 Jessica Davies and Fahiem Bacchus. Exploiting the Power of mip Solvers in maxsat. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *LNCS*, pages 166–181. Springer, 2013.

13 Jessica Davies and Fahiem Bacchus. Postponing Optimization to Speed Up MAXSAT Solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *LNCS*, pages 247–262. Springer, 2013.

14 Emir Demirovic, Nysret Musliu, and Felix Winter. Modeling and solving staff scheduling with partial weighted maxsat. *Annals OR*, 275(1):79–99, 2019. `doi:10.1007/s10479-017-2693-y`.

15 Niklas Eén and Niklas Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

16 Zhaohui Fu and Sharad Malik. On Solving the Partial MAX-SAT Problem. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 4121 of *LNCS*, pages 252–265. Springer, 2006.

17 Federico Heras, António Morgado, and João Marques-Silva. An empirical study of encodings for group maxsat. In Leila Kosseim and Diana Inkpen, editors, *Canadian Conference on Artificial Intelligence*, volume 7310 of *LNCS*, pages 85–96. Springer, 2012. `doi:10.1007/978-3-642-30353-1_8`.

18 Toshinori Hosokawa, Hiroshi Yamazaki, Kenichiro Misawa, Masayoshi Yoshimura, Yuki Hirama, and Masavuki Arai. A Low Capture Power Oriented X-filling Method Using Partial MaxSAT Iteratively. In *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 1–6. IEEE, 2019.

19 Alexey Ignatiev, António Morgado, and João Marques-Silva. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *International Conference on Theory and Applications of Satisfiability Testing*, volume 10929 of *LNCS*, pages 428–437. Springer, 2018. `doi:10.1007/978-3-319-94144-8_26`.

20 Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64, 2019. doi:10.3233/SAT190116.

21 Daniel Jackson. *Software Abstractions: Logic, language, and analysis.* MIT Press, 2006.

22 Mikoláš Janota, Inês Lynce, Vasco Manquinho, and Joao Marques-Silva. PackUp: Tools for Package Upgradability Solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):89–94, 2012.

23 Manu Jose and Rupak Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *Programming Language Design and Implementation*, pages 437–446. ACM, 2011.

24 Eunsuk Kang, Aleksandar Milicevic, and Daniel Jackson. Multi-representational security analysis. In *fse*, FSE 2016, pages 181–192. ACM, 2016.

25 Sarfraz Khurshid and Darko Marinov. Testera: Specification-based testing of java programs using SAT. *Autom. Softw. Eng.*, 11(4):403–434, 2004.

26 Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A Partial Max-SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012. doi:10.3233/sat190091.

27 Zhendong Lei, Shaowei Cai, Dongxu Wang, Yongrong Peng, Fei Geng, Dongdong Wan, Yiping Deng, and Pinyan Lu. Cashwmaxsat: Solver description. *MaxSAT Evaluation*, 2021:8, 2021.

28 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining clause learning and branch and bound for maxsat. In Laurent D. Michel, editor, *Principles and Practice of Constraint Programming*, volume 210 of *LIPIcs*, pages 38:1–38:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CP.2021.38.

29 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Boosting branch-and-bound maxsat solvers with clause learning. *AI Communications*, 35(2):131–151, 2022.

30 Ferney A. Maldonado-Lopez, Jaime Chavarriaga, and Yezid Donoso. Detecting network policy conflicts using alloy. In Yamine Aït Ameur and Klaus-Dieter Schewe, editors, *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, volume 8477 of *LNCS*, pages 314–317. Springer, 2014. doi:10.1007/978-3-662-43652-3_31.

31 Vasco Manquinho, Joao Marques-Silva, and Jordi Planes. Algorithms for Weighted Boolean Optimization. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 5584 of *LNCS*, pages 495–508. Springer, 2009.

32 João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Ann. Math. Artif. Intell.*, 62(3-4):317–343, 2011. doi:10.1007/s10472-011-9233-2.

33 João Marques-Silva and Jordi Planes. On Using Unsatisfiability for Solving Maximum Satisfiability. *CoRR*, 2007. arXiv:0712.1097.

34 Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental Cardinality Constraints for MaxSAT. In *Principles and Practice of Constraint Programming*, volume 8656 of *LNCS*, pages 531–548. Springer, 2014.

35 Ruben Martins, Vasco Manquinho, and Inês Lynce. On partitioning for maximum satisfiability. In *European Conference on Artificial Intelligence*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 913–914. IOS Press, 2012.

36 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: a Modular MaxSAT Solver. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 8561 of *LNCS*, pages 438–445. Springer, 2014.

37 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Community-based partitioning for maxsat solving. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *LNCS*, pages 182–191. Springer, 2013.

38 Erick Moreno-Centeno and Richard M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Oper. Res.*, 61(2):453–468, 2013. doi:10.1287/opre.1120.1139.

39 A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

40 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-Guided MaxSAT with Soft Cardinality Constraints. In *Principles and Practice of Constraint Programming*, volume 8656 of *LNCS*, pages 564–573. Springer, 2014.

41 António Morgado, Alexey Ignatiev, and João Marques-Silva. MSCG: robust core-guided maxsat solving. *J. Satisf. Boolean Model. Comput.*, 9(1):129–134, 2014. doi:10.3233/sat190105.

42 Miguel Neves, Ruben Martins, Mikolás Janota, Inês Lynce, and Vasco Manquinho. Exploiting resolution-based representations for maxsat solving. In Marijn Heule and Sean A. Weaver, editors, *International Conference on Theory and Applications of Satisfiability Testing*, volume 9340 of *LNCS*, pages 272–286. Springer, 2015.

43 Pedro Orvalho, Vasco Manquinho, and Ruben Martins. Upmax: User partitioning for maxsat, 2023. arXiv:2305.16191.

44 Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic polynomial watchdog encoding for solving weighted maxsat. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 10929 of *LNCS*, pages 37–53. Springer, 2018.

45 Marek Piotrów. UWrMaxSat – a new MiniSat+ – based Solver in MaxSAT Evaluation 2019. *MaxSAT Evaluation 2019 : Solver and Benchmark Descriptions*, B-2019-2:11, 2019.

46 Marek Piotrów. UWrMaxSat: Efficient Solver for MaxSAT and Pseudo-Boolean Problems. In *International Conference on Tools with Artificial Intelligence*, pages 132–136. IEEE, 2020.

47 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid maxsat solver. In *International Conference on Theory and Applications of Satisfiability Testing*, volume 9710 of *LNCS*, pages 539–546. Springer, 2016.

48 Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Starexec: A cross-community infrastructure for logic solving. In *International Joint Conference on Automated Reasoning*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014.

49 Caroline Trippel, Daniel Lustig, and Margaret Martonosi. CheckMate: Automated synthesis of hardware exploits and security litmus tests. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, 2018-Octob:947–960, 2018.

50 Allen Van Gelder. Variable independence and resolution paths for quantified boolean formulas. In *Principles and Practice of Constraint Programming*, volume 6876 of *LNCS*, pages 789–803. Springer, 2011. doi:10.1007/978-3-642-23786-7_59.

51 Rouven Walter, Christoph Zengler, and Wolfgang Küchlin. Applications of maxsat in automotive configuration. In *International Configuration Workshop*, volume 1128 of *CEUR Workshop Proceedings*, pages 21–28. CEUR-WS.org, 2013.

52 Robert A. Yates, Bertram Raphael, and Timothy P. Hart. Resolution graphs. *Artificial Intelligence*, 1(4):257–289, 1970. doi:10.1016/0004-3702(70)90011-1.

53 Aolong Zha, Miyuki Koshimura, and Hiroshi Fujita. N-level modulo-based CNF encodings of pseudo-boolean constraints for maxsat. *Constraints An Int. J.*, 24(2):133–161, 2019. doi:10.1007/s10601-018-9299-0.

54 Changjian Zhang, Ryan Wagner, Pedro Orvalho, David Garlan, Vasco Manquinho, Ruben Martins, and Eunsuk Kang. AlloyMax: bringing maximum satisfaction to relational specifications. In Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta, editors, *fse*, pages 155–167. ACM, 2021.

55 Lei Zhang and Fahiem Bacchus. MAXSAT heuristics for cost optimal planning. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 2012. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5190.

QMusExt: A Minimal (Un)satisfiable Core Extractor for Quantified Boolean Formulas

Andreas Plank
Johannes Kepler Universität Linz, Austria

Martina Seidl
Johannes Kepler Universität Linz, Austria

Abstract

In this paper, we present QMusExt, a tool for the extraction of minimal unsatisfiable sets (MUS) from quantified Boolean formulas (QBFs) in prenex conjunctive normal form (PCNF). Our tool generalizes an efficient algorithm for MUS extraction from propositional formulas that analyses and rewrites resolution proofs generated by SAT solvers.

In addition to extracting unsatisfiable cores from false formulas in PCNF, we apply QMusExt also to obtain satisfiable cores from Q-resolution proofs of true formulas in prenex disjunctive normal form (PDNF).

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Minimal Unsatisfiable Core, Quantified Boolean Formula

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.20

Supplementary Material *Software (Source Code)*: https://github.com/PlankAndreas/QMusExt
archived at swh:1:dir:29b2426d2111cd93ebe0c5a9c41e9dc1f337fcce

Funding This work has been supported by the LIT AI Lab funded by the State of Upper Austria.

1 Introduction

We present the tool QMusExt that computes a *minimal unsatisfiable set* (MUS), also called minimal unsatisfiable core, of a false quantified Boolean formula (QBF) $\Pi.\phi$ in prenex conjunctive normal form. An MUS is a subformula of $\phi' \subseteq \phi$ such that $\Pi.\phi'$ is false and removing any clause from ϕ' would make the formula true. Hence, an MUS describes a set of contradicting constraints from which no clause may be removed without eliminating the inconsistency as well. As this information is very important for understanding the reason of an inconsistency, many approaches have been presented to compute minimal unsatisfiable cores for propositional formulas [6]. In general, the MUS of a formula is not necessarily unique, a formula can have multiple MUSes and usually the smaller ones are preferred, i.e., size is a measure on the quality of the MUS extraction algorithm.

For QBFs, only few approaches for calculating MUSes exist so far, although MUS extraction is an important problem here as well. In [7, 6], theoretical properties of MUS extraction have been studied. An approach that extracts unsatisfiable cores which are not necessarily minimal is employed in [13] to validate the correctness of QBF solving results. In [4] the extraction of unsatisfiable cores is discussed in the context of the quantified MaxSAT problem. An approach to extract minimal unsatisfiable cores from false QBFs in PCNF was presented by Lonsing and Egly [8]. In this work, the solver DepQBF was equipped with an interface for incremental solving that provides a clause grouping feature. They showed that with this feature, the iterative clause set refinement approach with selector variables is straight-forward to implement for PCNF formulas. To the best of our knowledge, they provided the first available tool for MUS extraction. Most recently, Niskanen et al. presented an approach to find a smallest MUS of a false QBF based on implicit hitting sets [12].

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 20; pp. 20:1–20:10
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The approach implemented in our tool QMusExt works differently. It applies a proof-based approach that was originally suggested for propositional formulas [2]. In particular, it reduces the set of initial clauses of a Q-resolution proof as produced by modern QBF solvers until an MUS is extracted. Therefore, the proof needs to be iteratively updated. By exploiting the duality of true and false QBFs, satisfiable cores of true QBFs can be obtained by employing our approach. Our tool QMusExt is implemented in C and is available under MIT license at

`https://github.com/PlankAndreas/QMusExt`.

2 Preliminaries

We consider QBFs of the form $\Pi.\phi$, where $\Pi = Q_1X_1...Q_nX_n$ is called the *quantifier prefix*, $X_1, \dots, X_n$ are pairwise disjoint sets of Boolean variables, $Q_i \in \{\forall, \exists\}$, and $Q_i \neq Q_{i+1}$. The *matrix* ϕ is a propositional formula either in *prenex conjunctive normal form (PCNF)*, i.e., it is a conjunction of clauses, or in *prenex disjunctive normal form (PDNF)*, i.e., it is a disjunction of cubes. As usual, a *clause* is a disjunction of literals and a *cube* is a conjunction of literals. If convenient, we interpret clauses and cubes as sets of literals. A *literal* is a variable or a negated variable. We define $Var(l) = x$ if $l = x$ or $l = \bar{x}$ for any literal l. We say a literal is existential (universal), if its variable is existentially (universally) quantified. Further, $\bar{l} = x$ if $l = \bar{x}$ and $\bar{l} = \bar{x}$ if $l = x$. A quantifier prefix $Q_1X_1 \dots Q_nX_n$ imposes an ordering $<$ on the variables: if $x_i \in X_i$, $x_j \in X_j$, and $i < j$, then $x_i < x_j$. For a propositional formula ϕ, ϕ_l denotes the formula obtained by setting variable x to true if l $=$ x and by setting x to false if $l = \bar{x}$. A QBF $\forall X \Pi.\phi$ is true iff $\forall X' \Pi.\phi_x$ and $\forall X' \Pi.\phi_{\bar{x}}$ are true where $X' = X \setminus \{x\}$. Respectively, a QBF $\exists X \Pi.\phi$ is true iff $\exists X' \Pi.\phi_x$ or $\exists X' \Pi.\phi_{\bar{x}}$ is true. For example $\forall x \exists y.(x \vee y) \wedge (\bar{x} \vee \bar{y})$ is true and $\exists x \forall y.(x \vee y) \wedge (\bar{x} \vee \bar{y})$ is false. Every false QBF $\Pi.\phi$ in PCNF can be refuted by Q-resolution [5] which consists of the following three clause-derivation rules:

- **Axiom**: Any clause of ϕ can be derived.
- **Resolution**: From already derived clauses $C \vee x$ and $D \vee \bar{x}$, a clause $C \vee D$ can be derived if there is no literal l with $l, \bar{l} \in C \cup D$, $x \notin D$, $\bar{x} \notin C$, and x is existentially quantified.
- **Universal Reduction**: From an already derived clause $C \vee l$, a clause C can be derived if l is universal and there is no existential literal $k \in C$ with $l < k$.

A QBF is false iff the empty clause $\Box$ can be derived via Q-resolution. Dually, every true QBF $\Pi.\phi$ in PCNF can be proven by Q-resolution [3] which consists of the following three cube-derivation rules:

- **Axiom**: Let σ be a satisfying assignment of ϕ. Then cube $\bigwedge_{l \in \sigma} l$ can be derived.
- **Resolution**: From already derived cubes $C \wedge x$ and $D \wedge \bar{x}$, a cube $C \wedge D$ can be derived if there is no literal l with $l, \bar{l} \in C \cup D$, $x \notin D$, $\bar{x} \notin C$, and x is universally quantified.
- **Existential Reduction**: From an already derived cube $C \wedge l$, a cube C can be derived if l is existential and there is no universal literal $k \in C$ with $l < k$.

A QBF is true iff the empty cube can be derived via Q-resolution. A clause/cube derived via the resolution rule is called *resolvent*, while the parent clauses/cubes are called *antecedents*. Clauses with no antecedents are called *initial clauses*. Respectively, cubes without antecedents are called *initial cubes*. While initial clauses are directly available from the given PCNF formula, initial cubes have to be found by the QBF solver. Q-resolution proofs can be described in terms of *resolution graphs*. For a false QBF $\Pi.\phi$, a *resolution graph* $P = (V, E)$ is a directed acyclic graph (DAG). The set of vertices $V = V^{\mathsf{i}} \cup V^{\mathsf{d}}$ consists of initial clauses $V^{\mathsf{i}} \subseteq \phi$ and derived clauses V^{d}. Edges connect two antecedents and their resolvent, or a clause C and a clause C' that is obtained by universally reducing C. The only sink vertex is the empty clause denoted by $\Box$. Resolution graphs for true QBFs are defined respectively.

A vertex D is called *reachable* in resolution graph P from a vertex C iff there is a path from vertex C to vertex D. With $cone(P, C)$ we denote the set of all vertices reachable from vertex C in a resolution graph P (the *cone* of a clause C). Dually the set $unRe(P, C)$ contains all clauses not reachable from clause C. Finally, we define a resolution graph P to be *non-redundant* if all vertices are connected. In the following, we will use Q-resolution proofs to detect minimal (un)satisfiable cores which are defined as follows.

▶ **Definition 1** (Minimal Unsatisfiable Core). *For a false QBF $\Phi = \Pi.\phi$ in PCNF, a sub-formula $\phi' \subseteq \phi$ is a* minimal unsatisfiable core *of Φ, if $\Pi.\phi'$ is false and $\Pi.\phi' \backslash \{C\}$ is true for all $C \in \phi'$.*

The size of an unsatisfiable core is the number of its clauses. In general, minimal unsatisfiable cores are not unique and they can be of different size. For example, the formula $\exists x \forall y \exists z.((x \vee y \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{z}) \wedge (x \vee y \vee \bar{z}) \wedge (z \vee y))$ has minimal unsatisfiable cores $((x \vee y \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{z}))$, $((\bar{z}) \wedge (z \vee y))$, and $((x \vee y \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (x \vee y \vee \bar{z}))$.

▶ **Definition 2** (Minimal Satisfiable Core). *For a true QBF $\Phi = \Pi.\phi$ in PDNF, a sub-formula $\phi' \subseteq \phi$ is a* minimal satisfiable core *of Φ, if $\Pi.\phi'$ is true and $\Pi.\phi' \backslash \{C\}$ is false for all $C \in \phi'$.*

3 QMusExt

Our tool QMusExt extracts unsatisfiable cores from Q-resolution refutations of false QBFs. Further, it extracts satisfiable cores from Q-resolution satisfaction proofs of true QBFs. In both cases, QMusExt processes Q-resolution proofs in the QRP-format[1] and repeatedly calls the QBF solver DepQBF [9] in version 6.03 for deciding PCNF formulas and for producing proofs of modified formulas. In the following, we first introduce the algorithm implemented in QMusExt for extracting unsatisfiable cores, and then discuss the extraction of satisfiable cores.

Algorithm 1 Minimal Unsatisfiable Core Extraction.

```
Data: False QBF $\Pi.\phi$ in PCNF
Result: Minimal Unsat Core $V^{\mathsf{i}}$
(False, $P$) ← QBFSolver ($\Pi.\phi$) with $P = (V^{\mathsf{i}} \cup V^{\mathsf{d}}, E)$;
$P$ ← trim ($P$);
while unmarked clauses exists in $V^{\mathsf{i}}$ do
    $C^{\mathsf{i}}$ ← pickUnmarkedClause($V^{\mathsf{i}}$);
    (Val, $P'$) ← QBFSolver ($\Pi.unRe(P, C^{\mathsf{i}})$);
    if $Val == True$ then
        mark $C^{\mathsf{i}}$ as a MUS member;
    else
        $P''$ ← rebuildProof ($P$, $P'$);
        $P$ ← trim ($P''$);
    end
end
```

[1] http://fmv.jku.at/qbfcert/qrp.format

3.1 Basic Algorithm for the Extraction of Unsatisfiable Cores

Our tool QMusExt is based on the algorithm for the extraction of minimal unsatisfiable cores for propositional formulas presented in [2]. Whereas the original approach relies on propositional resolution proofs, QMusExt processes Q-resolution proofs. The latter contain not only applications of the axiom and the resolution rule, but also universal reductions. While this has some impacts on the implementation of QMusExt, conceptually the original algorithm is similar.

The approach implemented by QMusExt is summarized in Algorithm 1. The input is a false QBF $\Phi = \Pi.\phi$ in PCNF. First a QBF solver like DepQBF is called. We assume that the QBF solver returns a pair (`Val, `P) where `Val` is the truth value of the solved formula and $P = (V^{\mathsf{i}} \cup V^{\mathsf{d}}, E)$ is a Q-resolution refutation of Φ. In order to ensure that P is non-redundant, the function `trim` is called. Next, QMusExt checks if there is an unmarked clause in V^{i}. If a clause is unmarked it has not been checked so far if it belongs to the MUS. As long as there is one unmarked clause in V^{i}, such a clause C^{i} is selected. Then the QBF $\Pi.\phi'$ is solved where ϕ' consists of the clauses of P (initial and derived clauses) that are not in the cone of C^{i}, i.e., those clauses of P that are unreachable from C^{i}. If $\Phi.\phi'$ is true, then C^{i} is marked as MUS member. Otherwise, the solver returns a refutation P' of $\Pi.\phi'$. This proof does not contain C^{i} as initial clause, but P' could contain clauses from V^{d} as initial clauses which are not part of V^{i} and therefore also not part of the original clause set ϕ. Hence, it is not a proof of $\Pi.\phi$ in general. Based on information from proof P, P' can be modified to a proof P'' such that it contains only initial clauses from V^{i}. In consequence, P'' is a proof of $\Pi.\phi$. This proof P'' contains at least one initial clause less than P (clause C^{i}), but in many cases other clauses other than C^{i} from V^{i} are no initial clauses of P'' as well, because they are not needed to justify initial clauses from P'. In this way, multiple clauses not belonging to the MUS can be eliminated in one step. For the next iteration, P is replaced by a trimmed version of P'', i.e., all clauses that are not connected are removed to ensure that P is non-redundant. The approach is illustrated by the following example.

▶ **Example 3.** Consider the following QBF $\Phi_1 = \Pi.\phi$ which has seven clauses

$$\exists a, b \forall x, y \exists c, d.(\bar{b} \vee x \vee c) \wedge (b \vee x \vee c) \wedge (b \vee \bar{y} \vee \bar{c} \vee \bar{d}) \wedge (\bar{b} \vee y \vee d) \wedge (\bar{a} \vee x) \wedge (a \vee x \vee \bar{c} \vee \bar{d}) \wedge (a \vee x \vee c).$$

The resolution proof P_1 in Figure 1 witnesses that this formula is false. For convenience, the initial clauses V^{i} are labeled by C^{i}_j (with $1 \leq j \leq 7$) and the derived clauses V^{d} are labeled by C^{d}_k (with $1 \leq k \leq 10$). In this resolution graph with $V^{\mathsf{i}} = \phi$ all clauses are connected to the empty clause. Hence, it is already non-redundant.

In the first step, we remove clause $C^{\mathsf{i}}_1 = (\bar{b} \vee x \vee c)$ as well as its cone clauses $C^{\mathsf{d}}_6 = (x \vee c)$, $C^{\mathsf{d}}_8 = (x \vee \bar{y} \vee d)$, $C^{\mathsf{d}}_9 = (x \vee \bar{y})$, and $C^{\mathsf{d}}_{10} = \Box$. Those clauses are highlighted in Figure 1. Now a QBF solver is invoked on all remaining clauses, i.e., on the QBF $\Phi_2 = \Pi.V^{\mathsf{i}} \setminus \{C^{\mathsf{i}}_1\} \cup V^{\mathsf{d}} \setminus \{C^{\mathsf{d}}_6, C^{\mathsf{d}}_8, C^{\mathsf{d}}_9, C^{\mathsf{d}}_{10}\}$.

Also Φ_2 is false. Therefore, we can conclude that the clause $C^{\mathsf{i}}_1 = (\bar{b} \vee x \vee c)$ is not part of the minimal unsatisfiable core and can be removed permanently. The resolution proof P_2 of Φ_2 (the highlighted part of the proof shown in Figure 2 is not a resolution proof of Φ_1 as it contains initial clauses that do not occur in ϕ. However, the proof of Φ_1 can be used to rewrite the proof of Φ_2 such such that it becomes a proof of Φ_1 without using C^{i}_1. In particular, we need to introduce derivations for initial clauses of P_2 that are from V^{d}. These derivations are obtained from P_1. For example, the derivation for C^{d}_4 needs to be added. In the new proof, clause C^{i}_2 is not needed for proving Φ_1. Therefore, it is also not part of the MUS. The new proof has only five initial clauses. In the next five iterations, we find out that none of them can be removed, i.e., all of them are part of the MUS.

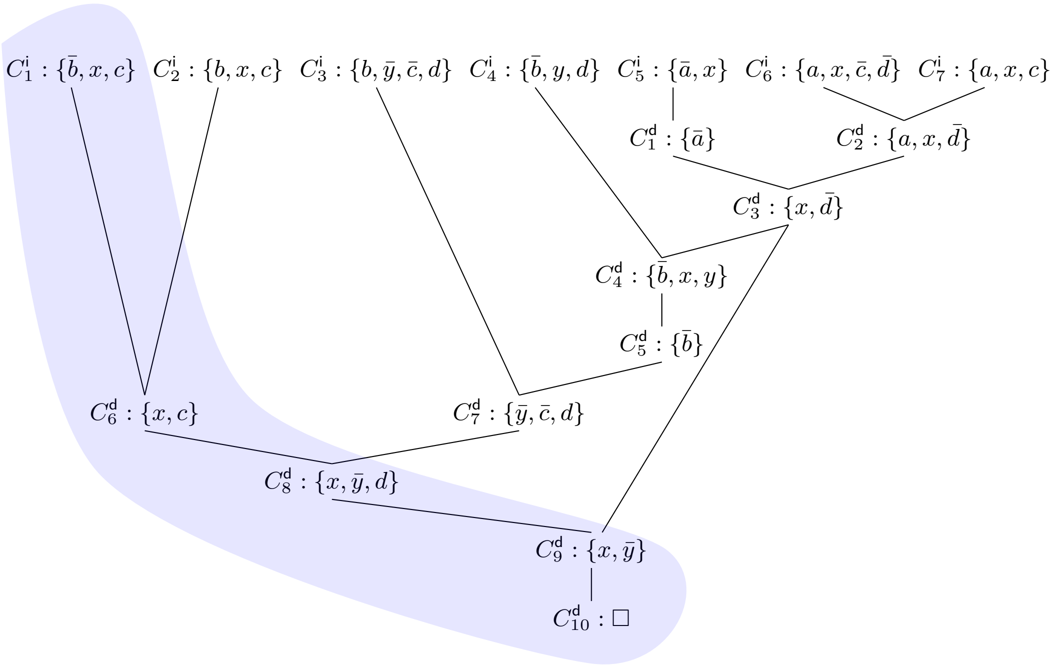

$C_1^{\mathsf{i}} : \{\bar{b}, x, c\}$
$C_2^{\mathsf{i}} : \{b, x, c\}$
$C_3^{\mathsf{i}} : \{b, \bar{y}, \bar{c}, d\}$
$C_4^{\mathsf{i}} : \{\bar{b}, y, d\}$
$C_5^{\mathsf{i}} : \{\bar{a}, x\}$
$C_6^{\mathsf{i}} : \{a, x, \bar{c}, \bar{d}\}$
$C_7^{\mathsf{i}} : \{a, x, c\}$
$C_1^{\mathsf{d}} : \{\bar{a}\}$
$C_2^{\mathsf{d}} : \{a, x, \bar{d}\}$
$C_3^{\mathsf{d}} : \{x, \bar{d}\}$
$C_4^{\mathsf{d}} : \{\bar{b}, x, y\}$
$C_5^{\mathsf{d}} : \{\bar{b}\}$
$C_6^{\mathsf{d}} : \{x, c\}$
$C_7^{\mathsf{d}} : \{\bar{y}, \bar{c}, d\}$
$C_8^{\mathsf{d}} : \{x, \bar{y}, d\}$
$C_9^{\mathsf{d}} : \{x, \bar{y}\}$
$C_{10}^{\mathsf{d}} : \square$

Figure 1 Initial resolution refutation for QBF Φ of Example 3 as returned by the QBF solver. Assume that the clause C_1^{i} is tested for its MUS membership. The highlighted clauses are in the cone of C_1^{i}. Only those clauses which are not highlighted are passed to the QBF solver in the next iteration.

3.2 Extraction of Satisfiable Cores

In contrast to SAT where only unsatisfiable formulas have resolution proofs, also true QBFs have resolution proofs. As QBFs are usually in PCNF, the solver has to provide initial cubes that are satisfying assignments of the matrix, i.e., for a true QBF $\Pi.\phi$ in PCNF, the solver provides a PDNF representation $\Pi.\psi$ from which the empty cube is derived by using the resolution rule and the existential reduction rule. We can now ask the question what is a minimal satisfiable core of $\Pi.\psi$? Our tool can directly answer this question by processing the Q-resolution satisfaction proof in a similar manner as discussed above. Minimal satisfiable cores might be used to find smaller proofs for true formulas. For true formulas, the clausal representation of the input formula is disadvantageous in general, leading to large initial cubes and large proofs. As an effect, the proofs are often very large and also the Skolem functions, the solutions that are extracted from the proofs according to approaches as presented in [1], are large as well.

4 Evaluation

In this section, we evaluate our tool QMusExt on false (true) instances to extract minimal unsatisfiable cores and minimal satisfiable cores. In our implementation we used hash maps as the data structure to store the resolution refutation. This design choice causes a slightly higher memory usage compared to arrays, however tests showed a significant speed up in computation time, due to efficient proof manipulations during the iterations. We also decided to closely interact with DepQBF via API calls, reducing the time needed for the required

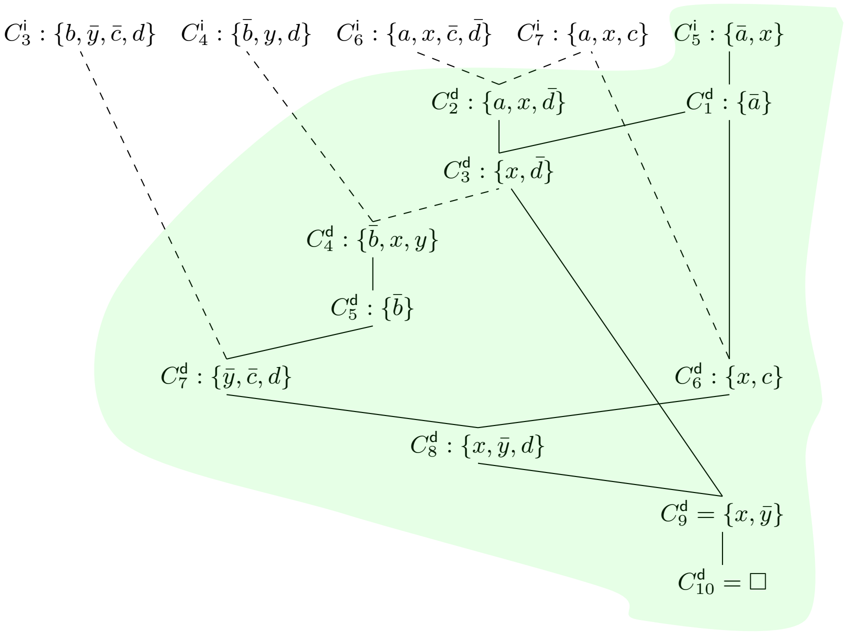

$C_3^{\mathsf{i}} : \{b, \bar{y}, \bar{c}, d\}$
$C_4^{\mathsf{i}} : \{\bar{b}, y, d\}$
$C_6^{\mathsf{i}} : \{a, x, \bar{c}, \bar{d}\}$
$C_7^{\mathsf{i}} : \{a, x, c\}$
$C_5^{\mathsf{i}} : \{\bar{a}, x\}$
$C_2^{\mathsf{d}} : \{a, x, \bar{d}\}$
$C_1^{\mathsf{d}} : \{\bar{a}\}$
$C_3^{\mathsf{d}} : \{x, \bar{d}\}$
$C_4^{\mathsf{d}} : \{\bar{b}, x, y\}$
$C_5^{\mathsf{d}} : \{\bar{b}\}$
$C_7^{\mathsf{d}} : \{\bar{y}, \bar{c}, d\}$
$C_6^{\mathsf{d}} : \{x, c\}$
$C_8^{\mathsf{d}} : \{x, \bar{y}, d\}$
$C_9^{\mathsf{d}} = \{x, \bar{y}\}$
$C_{10}^{\mathsf{d}} = \square$

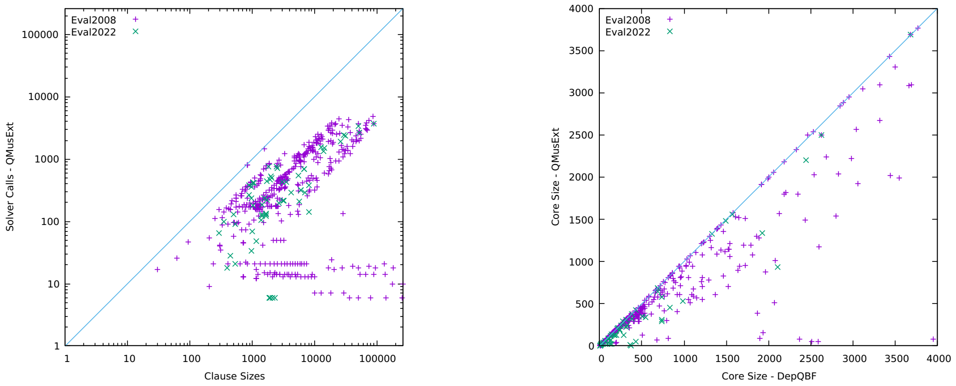
Figure 2 Rewritten resolution refutation after one iteration. The highlighted part is the proof for the formula that contains the clauses not reachable from C_1^{i} (the clauses not highlighted in the proof above). Hence, this is not a proof of Φ. The dashed edges and vertices from the matrix of Φ are added in order to obtain a proof for Φ. This proof does not include C_1^{i} as initial clause. Further, it does not include C_2^{i}.

solver calls. All experiments were run on a cluster of dual-socket AMD EPYC 7313 @ 16 × 3.7GHz machines with 4GB memory limit and 1800 seconds as timeout. All experimental data is available at the webpage of our tool.

4.1 Extraction of Unsatisfiable Cores

For MUS extraction, we consider the formulas of the PCNF track of the QBFEval 2022 and of the QBFEval 2008. All formulas are available at QBFLib.[2] To identify false formulas we run DepQBF [9] in standard configuration and selected all false formulas that could be solved within a time limit of 1800 seconds. Out of 1141 formulas of the eval2008 benchmark set (resp. 259 of the eval2022 benchmark set) 683 (resp. 137) formulas were found to be false. For the 2008 benchmarks, QMusExt could find the MUSes of 436 formulas with an average size of 533.50 clauses while the proofs contain 650.58 initial clauses and the original PCNFs 16903.04 clauses on average. For the 2022 benchmarks, QMusExt could find the MUSes of 62 formulas with an average size of 406.63 clauses while the proofs contain 500.53 initial clauses and the original PCNFs 12270.57 clauses on average. Hence we observe an decrease of 96.84 % and 96.68 % of used clauses compared to the initial clauses and a reduction in proof clauses of 18.00 % and 18.76 %. The reductions are summarized in Table 2. The average runtime for successful executions was 120.67 seconds for the 2008 benchmarks and 131.01 seconds for the 2022 benchmarks. On average 444.70 and 265.81 solver calls were needed

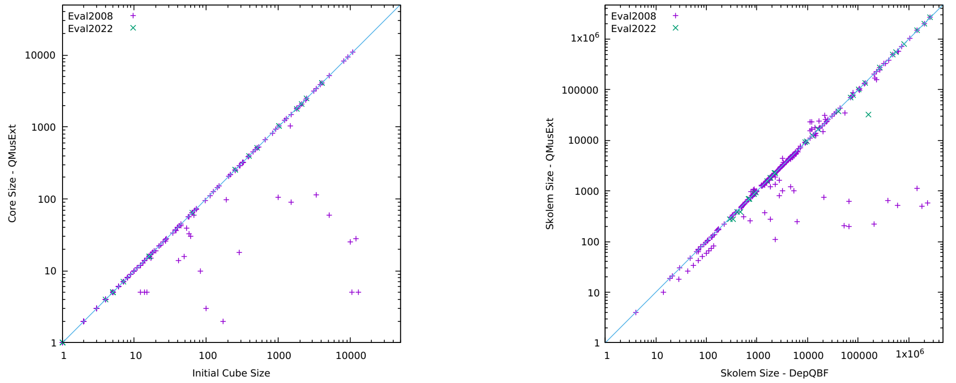
[2] http://www.qbflib.org



(a) Number of solver calls per formula within QMusExt.

(b) Comparison of MUS sizes produced by DepQBF vs. MUS sizes produced by QMusExt.

Figure 3 Results for false formulas.



(a) Size reduction of the initial cubes from Q-resolution satisfaction proofs.

(b) Size comparison of Skolem functions from initial proof vs Skolem functions from satisfiable core proofs.

Figure 4 Results for true formulas.

to find MUSes. Figure 3a relates number of solver calls and clause sizes. In the worst case, only one clause is eliminated per iteration, i.e., the approach is linear in the formula size. In practice, fewer calls are need indicating the scalability of the approach.

We compared QMusExt to the approach implemented in DepQBF [8] and the recent approach SMUSer [12] that computes minimum unsatisfiable cores, i.e., a MUS with the smallest possible cardinality. As finding the smallest MUSes is a computationally harder problem than finding any MUS, it is not surprising that SMUSer finds fewer MUSes compared to the other two approaches within the given time limit. In particular, it finds the smallest MUS for 32 formulas form the 2008 benchmark set. For the 2022 benchmark set, we did not obtain any result from SMUSer. For all of the formulas for which SMUSer could find a result, also QMusExt and DepQBF found MUSes. For 24 of these, our tool found cores of the same size as the cores found by SMUSer. The others differ by 18 clauses at most. Table 1 summarizes the number of solved instances and the average core sizes.

DepQBF is able to find MUSes of 679 (benchmarks from 2008) and of 134 (formulas from 2022). It is not surprising that DepQBF finds more MUSes with the incremental approach than QMusExt, although QMusExt also relies on DepQBF internally. For the algorithm

Table 1 Comparison of the evaluation results.

	solved instances				average core size			
	FALSE		TRUE		FALSE		TRUE	
	eval08 683 inst.	eval22 137 inst.	eval08 458 inst.	eval22 122 inst.	eval08	eval22	eval08	eval22
QMusExt	436	62	253	31	534	407	316	589
DepQBF	679	134	–	–	2438	1695	–	–
SMUSer	32	–	–	–	146	–	–	–

implemented in QMusExt, proofs have to be generated, analyzed and rewritten. If proof generation is enabled, certain pruning techniques have to be disabled slowing down the solving process. Further, the proof size can be very large, requiring the implementation of efficient hashing techniques for finding nodes in the resolution graph. When we compare the sizes of the MUSes produced by DepQBF to the sizes of the MUSes produced by QMusExt as done in Figure 3b we see that the cores found by QMusExt are of equal size or smaller.

4.2 Extraction of Satisfiable Cores

We also applied QMusExt on true formulas and observe it it can also find minimal satisfiable cores of the PDNF. For our experiments, we selected those formulas from the 2022 formulas and, respectively, from the 2008 formulas, which could be solved by DepQBF in 1800 seconds. Out of 458 (122) true formulas, our tool could find satisfiable cores for 253 and, respectively, 31 formulas. Out of these, 28 could by decreased in average by 61.32 %. Figure 4a shows the reduction for the individual formulas. In the most extreme case, a PDNF with 10096 cubes could be reduced to a PDNF with 25 cubes. We also calculated the Skolem functions from the original set of initial cubes as well as from the formula reduced to a minimal satisfiable core. The result is shown in Figure 4b. The Skolem functions are extracted with the QBFCert framework [11] and represented as And-Inverter Graphs in the Aiger format.[3] We measure the size in terms of gate numbers. In some cases, we observe a slight increase in the size of the Skolem functions, while there are also cases where the size could be considerably decreased. In the most extreme case, the Skolem function could be reduced by 99.98 %. Details are summarized in Table 1 and Table 2.

Table 2 Average size of cores generated by QMusExt (core), average formula sizes (formula), average number of axiom clauses/cubes (proof), reductions when applying QMusExt and average run time of QMusExt.

		avg. size			reductions		
		formula	proof	core	formula size	proof	avg. run time (s)
FALSE	eval08	651	16904	534	96.84%	17.97%	120.67
	eval22	501	12271	407	96.68%	18.76%	131.01
TRUE	eval08	840	17950	316	98.24%	62.38%	439.11
	eval22	595	70213	589	99.16%	1.01%	179.15

[3] http://fmv.jku.at/aiger/

5 Conclusion

We presented QMusExt, the first tool that implements the extraction of unsatisfiable cores of false QBFs based on Q-resolution proofs. Originally, the approach was successfully applied for SAT [2]. Our experiments indicate that the approach is also promising for QBFs. In particular, we could observe that the number of necessary solver calls is smaller than the number of clauses of the input formula. Not surprisingly, an approach based on selector variables implemented with the incremental interface of the QBF solver DepQBF is more efficient in terms of runtime. However, the approach of QMusExt finds smaller unsatisfiable cores in many cases. Further, due to the duality of false and true QBFs, the tool can be be applied for the extraction of satisfiable cores from PDNF formulas as produced by solvers as well.

In the future we plan to adapt optimizations of the basic algorithm as proposed in [10] for QBFs and combine Q-resolution based approaches with approaches based on selector variables. In addition, we further plan to investigate the pruning potential of proofs and function extraction.

This work has been supported by the LIT AI Lab funded by the State of Upper Austria.

References

1 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods Syst. Des.*, 41(1):45–65, 2012. doi:10.1007/s10703-012-0152-6.

2 Nachum Dershowitz, Ziyad Hanna, and Alexander Nadel. A scalable algorithm for minimal unsatisfiable core extraction. In *Proc. of the 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2006)*, volume 4121 of *Lecture Notes in Computer Science*, pages 36–41. Springer, 2006. doi:10.1007/11814948_5.

3 Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Clause/term resolution and learning in the evaluation of quantified boolean formulas. *J. Artif. Intell. Res.*, 26:371–416, 2006. doi:10.1613/jair.1959.

4 Alexey Ignatiev, Mikolás Janota, and João Marques-Silva. Quantified maximum satisfiability. *Constraints*, 21(2):277–302, 2016. doi:10.1007/s10601-015-9195-9.

5 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995. doi:10.1006/inco.1995.1025.

6 Hans Kleine Büning and Oliver Kullmann. Minimal unsatisfiability and autarkies. In *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 571–633. IOS Press, 2021.

7 Hans Kleine Büning and Xishun Zhao. Minimal false quantified boolean formulas. In Armin Biere and Carla P. Gomes, editors, *Proc. of the 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2006)*, volume 4121 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 2006.

8 Florian Lonsing and Uwe Egly. Incrementally computing minimal unsatisfiable cores of qbfs via a clause group solver API. In *Proc. of the 18th Int. Conf. on Theory and Applications of Satisfiabily Testing (SAT 2015)*, volume 9340 of *Lecture Notes in Computer Science*, pages 191–198. Springer, 2015. doi:10.1007/978-3-319-24318-4_14.

9 Florian Lonsing and Uwe Egly. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In *Proc. of the 26th Conf. on Automated Deduction (CADE 26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 371–384. Springer, 2017. doi:10.1007/978-3-319-63046-5_23.

10 Alexander Nadel. Boosting minimal unsatisfiable core extraction. In *Proc. of 10th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD 2010)*, pages 221–229. IEEE, 2010. URL: https://ieeexplore.ieee.org/document/5770953/.

11 Aina Niemetz, Mathias Preiner, Florian Lonsing, Martina Seidl, and Armin Biere. Resolution-based certificate extraction for QBF – (tool presentation). In *Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2012)*, volume 7317 of *Lecture Notes in Computer Science*, pages 430–435. Springer, 2012.

12 Andreas Niskanen, Jere Mustonen, Jeremias Berg, and Matti Järvisalo. Computing smallest muses of quantified boolean formulas. In *Proc. of the 16th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022)*, volume 13416 of *Lecture Notes in Computer Science*, pages 301–314. Springer, 2022. doi:10.1007/978-3-031-15707-3_23.

13 Yinlei Yu and Sharad Malik. Validating the result of a quantified boolean formula (QBF) solver: theory and practice. In *Proc. of the 2005 Conf. on Asia South Pacific Design Automation, (ASP-DAC 2005)*, pages 1047–1051. ACM Press, 2005.

Faster LRAT Checking Than Solving with CaDiCaL

Florian Pollitt
Universität Freiburg, Germany

Mathias Fleury
Universität Freiburg, Germany

Armin Biere
Universität Freiburg, Germany

Abstract

DRAT is the standard proof format used in the SAT Competition. It is easy to generate but checking proofs often takes even more time than solving the problem. An alternative is to use the LRAT proof system. While LRAT is easier and way more efficient to check, it is more complex to generate directly. Due to this complexity LRAT is not supported natively by any state-of-the-art SAT solver. Therefore Carneiro and Heule proposed the mixed proof format FRAT which still suffers from costly intermediate translation. We present an extension to the state-of-the-art solver CaDiCaL which is able to generate LRAT natively for all procedures implemented in CaDiCaL. We further present Lrat-Trim, a tool which not only trims and checks LRAT proofs in both ASCII and binary format but also produces clausal cores and has been tested thoroughly. Our experiments on recent competition benchmarks show that our approach reduces time of proof generation and certification substantially compared to competing approaches using intermediate DRAT or FRAT proofs.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases SAT solving, Proof Checking, DRAT, LRAT, FRAT

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.21

Supplementary Material
Dataset (Log files): `https://cca.informatik.uni-freiburg.de/lrat/` [20]

Acknowledgements We thank reviewers of SAT'23 and MBMV'23 for their detailed comments as well as Mario Carneiro for making FRAT-rs publicly available.

1 Introduction

Proof production became an essential part in SAT solving. For instance, unsatisfiable problems only count as solved in the SAT Competition if a certifiable proof is provided. Proofs do increase trust in solving results by providing certificates that can be checked independently. To increase trust even further proof checkers can also be entirely verified [6, 16].

In the past the only format allowed in the SAT Competition was DRAT [23], even though the SAT Competition 2023 announced to allow additional formats. However, checking DRAT proofs often takes several times the amount of solving time. The problem with DRAT is that the format is not detailed enough to avoid search during checking. Both the solver and the checker have to propagate clauses (actually using similar data structures). To reduce this overhead (and simplify verification) all verified proof checkers expect an enriched format. The DRAT proof is augmented and converted by an (untrusted) external program into such an enriched format, e.g., LRAT [6] or GRAT [16], which contains enough information to avoid search and can then be checked easily by the verified proof checker.

On top of the actual clause contents (its literals) the LRAT [6] format requires the following additional information: (i) clause identifiers (ids) are used to reference clauses and to make clause deletion steps more concise; (ii) clause antecedent ids used in the resolution chain when deriving an added clause through reverse unit propagation (RUP) [12], i.e., as

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 21; pp. 21:1–21:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

asymmetric tautology (AT) [14]; (*iii*) the ID and further resolution paths to refute the resolvent of the added clause with all clauses containing a RAT (blocking) literal in case the added clause relies on the stronger resolution asymmetric tautology (RAT) property [15].

These RAT literals would be needed to model more powerful reasoning (such as blocked clause addition or symmetry breaking etc.) but neither our SAT solver CaDiCaL [4] nor any top performing SAT solver in the SAT Competition over the last 2 years actually used such reasoning. Therefore, our efforts to extend CaDiCaL did not need to address the full power of RAT and we can focus on producing "LRUP" proofs, i.e., reverse-unit-propagation (RUP) proofs, but still need to augment these proofs with ids and resolution chains.

A similar attempt [1] by Carneiro and Heule led to a new proof format, FRAT, that sits between LRAT (because it allows for justifications) and DRAT (because it still allows steps without justification). Their aim was to fill out most "*gaps*" and leave *"harder"* to implement cases as black box to be filled in by an (untrusted) proof checker, i.e., by their FRAT-rs tool used to convert an FRAT proof to a fully justified LRAT proof. In a recent paper [18] this limitation of the FRAT producing CaDiCaL [1] forced a parallel proof-producing version of the award winning SAT solver Mallob to deactivate all steps not covered by FRAT, i.e., most inprocessing, as native LRAT proof generation is needed.

In this tool paper, we present an extension of our SAT solver CaDiCaL [4] to generate the richer LRAT format directly. Our focus is on three different aspects: (A) producing LRAT proofs for all solver configurations on all benchmarks, (B) comparable performance and, further, (C) making sure the solver behaves the same with/without proof generation.

Our goal (A) lead us to reimplement LRAT generation in the conflict analysis and all inprocessing techniques of CaDiCaL, some of which were not covered in the FRAT [1] producing implementation, such as equivalent literal subsumption (Section 3).

Like other SAT solvers, CaDiCaL generates a vast number of proof steps from which at the end, a significant fraction turns out to be unnecessary for the derivation of the empty clause. Thus most tools that process DRAT or FRAT will trim these unnecessary steps from the proof. However, we are not aware of a tool that does this for LRAT. Therefore we implemented a new tool called Lrat-Trim to trim proofs down and improve the performance of checking the proof with the verified checker Cake_Lpr [21] (Section 4).

To validate robustness of our approach we extended CaDiCaL to internally check LRAT proofs too and fuzzed the extended solver. This allowed us to use the model-based tester Mobical (which comes with CaDiCaL) to find, debug, and fix bugs much more efficiently. We further ran the extended new solver on the unsatisfiable problems from the SAT Competition 2022. We observed (almost) no slow-down without proof production (0.3%) and only a small slow-down for producing LRAT (5%). Proof checking performance was improved considerably compared to the two competing approaches DRAT and FRAT (see Section 5). Checking (and producing) our LRAT proofs has an overhead of 30% over pure solving, compared to 125% for FRAT and 180% in the SAT Competition mode (i.e., slower than producing them). Without negligible overhead over plain solving with CaDiCaL, we managed to check proofs faster than they are produced for a state-of-the-art SAT solver.

Our CaDiCaL extension is available at `https://github.com/florianpollitt/radical` and will shortly be merged into the main CaDiCaL repository. Note that a preliminary version of this paper was presented at the MBMV workshop [19] as work in progress. Compared to that shorter version, we have improved and present Lrat-Trim, give an extensive evaluation on the entire problem set of the SAT Competition 2022 (not just a single problem) and in general provide more details on the implementation.

2 Preliminaries

For an introduction to SAT solving please refer to the *Handbook of Satisfiability* [5]. In our context it is sufficient to recall that SAT solvers build a partial assignment and along the way learn new clauses preserving satisfiability until either the assignment satisfies all clauses or the empty clause is derived, meaning that the problem is unsatisfiable.

A DRAT [23] proof is the sequence of all clauses learned (or in general deduced) by the SAT solver interleaved with clause deletion steps, which are used to help the proof checker to focus on the same clauses the solver would see at this point of the proof. This design principle helps DRAT [23] to easily capture all techniques currently used by SAT solvers without the need to provide more complex justification e.g. in the form of resolution chains.

The LRAT [6] proof format has more detailed information: Each clause is associated with a clause identifier and claimed to be the result of resolving/propagating several clauses in the given order. The list of antecedent clause ids forms a justification and is part of such an addition step in LRAT. In the rest of the paper we focus on finding these justification.

3 Implementation

The LRAT extension to CaDiCaL was implemented by the first author as part of his master project and proceeded in four stages: First, the internal proof checker in CaDiCaL for DRAT clauses was extended to produce LRAT proofs, which is quite inefficient but can still be enabled through the `--lrat-external` option. Second, a separate internal LRAT checker was added to CaDiCaL to validate proofs on-the-fly while running the solver. Third, we implemented LRAT production for CaDiCaL without any inprocessing. Finally, all different inprocessing techniques were instrumented to generate LRAT proof chains directly. Thanks to the second stage, proofs could be validated on-the-fly, dramatically reducing the implementation effort (particularly for debugging). The implementation of these four stages took around two months in total but the last two stages only two weeks.

The resolution chain for justifying a new clause can be computed alongside normal CDCL search with little computational overhead but clause minimization and shrinking are a bit more involved (Section 3.1). Proof production in preprocessing and inprocessing were of varying degree of difficulty. The most interesting inprocessing technique from this point of view is equivalent literal substitution which we discuss in Section 3.2.

3.1 Conflict Analysis

Most clauses derived by a SAT solver originate from clauses learned during conflict analysis. When the solver finds a mismatch between the current partial assignment and the clauses, i.e., a conflicting clause which is falsified, then this conflict is analyzed and a clause is learned which forces the solver to adjust the partial assignment. In the standard implementation of conflict analysis the learned clause is derived by resolving individual reason clauses in reverse assignment order, starting with the conflicting clause, which in turn immediately gives the necessary justification for the (non-minimized first UIP [24]) learned clause.

We have adapted our code to generate chains for various technique relying on conflict analysis such as hyper binary resolution [13] and vivification [17]. It is crucial to distinguish between techniques that eliminate false literals (thus, necessitating an extension of the proof chain) and those that do not.

One recent addition to improve conflict analysis is the concept of "shrinking" [9, 10] which can be interpreted as a more advanced version of "minimization" [8]. Minimization only removes literals from the learned clause following resolution paths in the implication graph, but does not add any literals. The additional idea in shrinking is to continue trying to resolve literals on a particular decision level until all but one (the first UIP on that level) is left, however, without being allowed to add literals from a lower decision level.

Our approach differs from the FRAT flow [1]. Their solver performs a post-process analysis of the final learned clause $C_{mini+shrink}$ to rediscover the necessary propagation by traversing the implication graph, which repeats conflict analysis work. In contrast, we split the justification process into two parts. First, we derive the justification for the clause C_{UIP} alongside conflict analysis with little to no overhead. Then, we derive the missing resolution steps between C_{UIP} and the shrunken and minimized clause $C_{mini+shrink}$ as a post-process analysis. We identify literals that differ and add the required reason clauses. Although we still traverse parts of the implication graph, we avoid repeating the conflict analysis.

Our Algorithm 1 shows the postprocessing step only. The first step has already derived the justification Chain_{UIP} for the first UIP clause $C_{original}$ from conflict analysis. Our postprocessing step calculates the justification chain in $\mathit{Chain}_{mini+shrink}$. For each removed literal L (in $C_{original}$ but not in $C_{shrunken}$), we extend the chain with additional justification steps (Line 3).

The function calculate_LRAT_Chain(L) (Line 5) extends the chains with the required reason and preserves the resolution order. It goes recursively over all literals of the reasons and extends the chain with the reason. If the function reaches a previously used reason (*already_added*), it can stop the analysis to avoid duplicated reasons in the chain. Our calculation stops when we reach literals that appear in $C_{shrunken}$ ($L \notin \mathit{Chain}_{new}$). After calculating the justification chain for minimization and shrink, we merge the two chains Chain_{UIP} and Chain_{new} (Line 4). Starting with an empty chain provides a valid proof when removing unit literals during both phases.

Algorithm 1 Recursively calculating the prefix LRAT chain for shrinking and minimizing.

```
   Data: currently build LRAT chain $Chain_{UIP}$
   Data: the clause before $C_{original}$ and after minimization and shrinking $C_{shrunken}$
   Result: resulting LRAT chain $Chain_{full}$

 1 foreach literal $L$ in $C_{original}$ do
 2     if $L$ not in $C_{shrunken}$ then
 3         calculate_LRAT_Chain($L$)

 4 $Chain_{full} := Chain_{mini+shrink} + Chain_{UIP}$

 5 calculate_LRAT_Chain (Literal $K$)
 6     $C$ := reason of $K$ in the current assignment
 7     foreach Literal $L$ in $C$ different from $K$ do
 8         $already\_added$ := reason of $L$ in $Chain_{mini+shrink}$
 9         if $\neg already\_added$ and $L \notin C_{shrunken}$ then
10             calculate_LRAT_Chain($L$)
11     append $C$ to $Chain_{mini+shrink}$
```

Our approach can potentially lead to duplicated unit clauses: We add unit clauses to the chain during conflict analysis. We can guarantee no duplicates here, but the same unit clause might also be added during post process analysis, which means it is actually needed earlier in $\mathit{Chain}_{mini+shrink}$ and we could remove it from Chain_{UIP}. Note that this cannot happen for larger clauses since they can appear at most once as a reason for some assignment. Since removing these unit clauses afterwards would be rather costly, we actually collect unit clauses separately and put them at the start of the merged chain after the post process analysis for $C_{shrunken}$ is finished. Like this, we can avoid duplicates and still get a correct justification chain for $C_{shrunken}$.

3.2 Equivalence Literal Substitution

While the justification process for clauses derived during variable elimination and other preprocessing techniques that rely on propagation and conflict analysis is similar to normal learning, producing LRAT proof justifications for equivalent literal substitution [5] is more involved.

Equivalent literal substitution detects and replaces equivalent literals by a chosen representative. For example, if the problem includes the three clauses $(\neg A \vee B)$, $(\neg B \vee C)$ and $(\neg C \vee A)$ we know that A, B and C are equivalent and we can replace all occurrences of either literal by one of the others. As is common we use Tarjan's algorithm [22] to detect cycles in the graph spanned by the binary clauses (i.e., the binary implication graph) and fix a representative for each cycle [5]. In the DRAT proof we can simply dump all changed clauses and delete the old ones.

For LRAT we have to produce the resolution chains. After fixing representatives, proof chains have to be produced for every changed clause separately. We derive the justification for each changed or removed literal, similarly as for the shrunken clause in conflict analysis 3.1.

Fixing the representative is a rather arbitrary choice (the smallest absolute value in this implementation). We considered changing this to the first visited literal during DFS in Tarjan's Algorithm, in order to allow reusing some computation and potentially shorten proofs, but in the end decided against changing solver behavior.

4 Trimming LRAT proofs

In preliminary experiments we observed that the FRAT flow [1] produced significantly smaller proofs. FRAT-RS trims the proof during translation to LRAT, i.e., it omits clauses that are not needed to derive the empty clause, allowing for much more efficient proof checking. We concluded that we needed a tool to do such trimming on LRAT directly in order to obtain an efficient pure LRAT proof generation and checking flow.

Even though trimming is effective, it is not obvious how to cheaply achieve such reduction for DRAT proofs because dependencies between proof steps are lacking. Luckily, in LRAT these dependencies are explicit. Therefore we implemented LRAT-TRIM [2], an open-source LRAT proof trimming and checking tool. It often reduces proofs by a factor of 2 to 3, again emphasizing how many useless clauses a SAT solver actually derives during search.

Trimming LRAT proofs consists of a backward reachability analysis starting from the empty clause towards the clauses of the original CNF, marking reached clauses as needed. Clauses unmarked after this traversal are redundant and can be trimmed. This algorithm is implemented by depth first search (DFS) along antecedent clauses in justification chains.

It also determines the last usage of each clause ID and remaps original clause ids to a consecutive ID range. On completion we can dump the proofs back to a file in a forward manner, only writing needed clauses and their antecedents and skipping redundant clauses. While doing this we can eagerly mark clauses once they are not used anymore.

Before starting to write proof lines, we check whether there are redundant original clauses and if so write a single deletion line with all unused original clause ids. This minimizes the life-span of clauses in the trimmed LRAT proof, both for added and original clauses. Note that LRAT-TRIM, in contrast to DRAT-Trim, does not require access to the original CNF nor looks at literals of clauses to trim proofs.

We also implemented a checking mode in LRAT-TRIM which, given the original CNF and an LRAT proof, checks that the resolution chains of added clauses can be resolved to produce the claimed clauses. It also checks that clauses are not used after they are deleted in a deletion step. This checking mode comes in two flavors. The default is to first trim the clauses with the trimming algorithm described above and only check needed clauses. Alternatively LRAT-TRIM supports forward checking, which checks added clauses on-the-fly during parsing and in particular allows to delete clauses in deletion steps eagerly.

On the one hand, forward checking reduces maximum memory usage to at most that of the solving process, whereas backward checking needs to keep the whole proof in memory which is usually much more than maximum usage during solving. On the other hand, forward checking substantially increases checking time, as all clauses have to be checked without trimming information, irrespective of being needed or not.

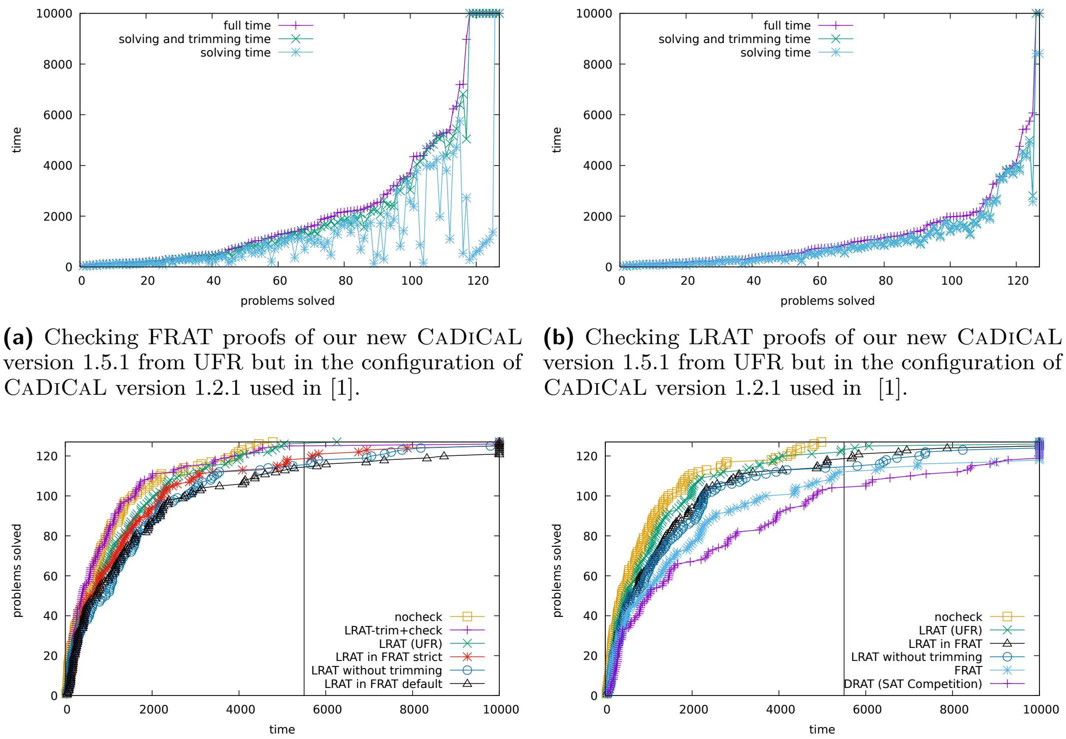
During the development of LRAT-TRIM substantial effort went into making parsing as fast and robust as possible and also provide meaningful error messages during parsing and checking. The parsing code amounts to roughly 900 lines of C code out of 2400 lines for the whole tool (including comments but formatted with CLANGFORMAT).

All three proof formats (DRAT, FRAT and LRAT) have a binary version. We implemented the binary format for LRAT (both in CADICAL and in LRAT-TRIM) which is only supported by CLRAT [7], a formally verified checker for LRAT using ACL2. We are grateful to Peter Lammich who provided us a tool that converts LRAT proofs (with some extra requirements on proofs) to GRAT [16] that his checker can check. However, GRAT is stricter as duplicate or extraneous ids are not allowed. We leave it to future work to produce stricter proofs.

5 Experiments

While checking for our extensions not to change solver behavior with and without proof generation, i.e., validating (C), we realized that two changes to the solver became necessary. First, scheduling of garbage collection during bounded-variable elimination depends on the number of bytes allocated for clauses, which changed with LRAT proof generation, as clauses require an ID and thus became larger. Therefore, our CADICAL extension always uses clause ids, which is not expected to have major impact on performance nor memory usage. The second change is due to the way conflicts were derived in equivalent literal detection. Originally detection was aborted on such a conflict, which we now simply delay until detection finishes. Then the conflicting literal is propagated to yield a proper LRAT proof.

Our goal (A) of being able to always generate correct proofs was tested by intensive fuzzing of our solver, proof generation, and proof checking. We attempted to apply the same approach to the FRAT extension of CADICAL [1] but immediately experienced failing proofs, due to several reasons, particularly with respect to handling unit clauses in the input



(a) Checking FRAT proofs of our new CaDiCaL version 1.5.1 from UFR but in the configuration of CaDiCaL version 1.2.1 used in [1].

(b) Checking LRAT proofs of our new CaDiCaL version 1.5.1 from UFR but in the configuration of CaDiCaL version 1.2.1 used in [1].

(c) Comparing trimmers Lrat-Trim vs. FRAT-rs (LRAT in FRAT) using CaDiCaL 1.5.1 proofs before checking with Cake_Lpr, except for the run "LRAT-trim+check" which checks with Lrat-Trim.

(d) CDF of all the solving and checking flows with the vertical black line indicating the 5,000 seconds timeout used for the solver, showing that our flow is the fastest with fewer timeouts.

Figure 1 Performance on unsatisfiable instances from the SAT Competition 2022.

CNF. We also observed that chains often listed the same clause id multiple times. Reducing these occurrences might lead to a substantial speedup, since justifying one literal can pull in several more clauses (e.g., if some of the literals have been removed by minimization).

After fuzzing, we ran our LRAT flow on the problems of the SAT Competition 2022 and found three issues: (*i*) Cake_Lpr did not accept some input files, because they contained trailing empty lines, which we then removed manually; (*ii*) Cake_Lpr requires a very large amount of memory (around the size of the proof file); (*iii*) one node of the cluster showed irregular behavior, when many proofs were written to the temporary disk at the same time, which lead to corrupted proof files resulting in an Lrat-Trim error. Reducing the number of jobs per node fixed this issue and we did not discover any further problem with the generated proofs, validating (A) and showing again the effectiveness of fuzzing.

To compare performance, i.e., showing that we achieved (B), of our extended version to the base version of CaDiCaL (added clause ids taking up space without being used), we let both versions write generated proofs to `/dev/null` in order to ensure that we do not introduce any bias due to file I/O limits as LRAT proofs exceed DRAT proofs in size substantially. This yielded an average overhead of 5% for our new LRAT proof production versus DRAT in base CaDiCaL.

For the remaining empirical analysis we have chosen to focus on the 127 benchmarks from the SAT Competition 2022, which were shown to be unsatisfiable during the competition. First, we tried to determine how much proofs can be reduced with our new tool Lrat-Trim.

It turns out, that some proofs were reduced to *one percent*, i.e., 99% of the output is not useful for deriving the contradiction. These problems stem from the `sudoku-N30` family. In other proofs 80% and more clauses are needed – most of these problems have a short runtime (around 200 s), contain a large amount of fixed variables and accordingly many clauses are simplified by removing these units, where each removal contributes a proof step.

In order to determine the performance of our new solving and checking flow, we compared the following three workflows: (*i*) the (competition) DRAT workflow, i.e., generating the DRAT proof, converting it to LRAT with DRAT-Trim, then checking that proof; (*ii*) the FRAT workflow, i.e., generating the FRAT proof, converting it to LRAT with FRAT-RS, then checking it; (*iii*) our new LRAT flow including generating, trimming, and checking the proof. All workflows use binary proof formats, except for feeding CAKE_LPR at the end.

We also ported the FRAT extensions [1] to the newest CADICAL version, but did not try to fix any issues. Nevertheless, we ran the ported version (see Figure 1a) which is now able to use the latest heuristics used in CADICAL, except for shrinking which had to be deactivated as it is not supported by the original FRAT code [1].

The first observation we can make is that the overhead of trimming and proof checking is quite consistent among our configurations, but wildly differs for FRAT: If many clauses without justification are used for the proof, the translation needs a lot of search – although, as expected, less than using the conversion to DRAT

To our surprise, we observed several timeouts though. They all seem to origin from one family submitted by AWS in 2022, where solving took less than 600 s, but elaboration (translation) never finishes. In comparison, DRAT-Trim also needs a very long time (6 000 s), but stays well below the time limit. It is unclear what the problem is and thus we tested one instance `aws-c-common:aws_priority_queue_s_sift_either` on a (twice as fast) computer where it took nearly 10 h to convert the 400 MB FRAT proof to a 3.8 GB LRAT proof. We have reported the issue on GitHub,[1] but have not heard back yet.

A comparison of LRAT-TRIM with FRAT-RS in both *normal mode* and *super strict mode* is shown in Figure 1c. We used the feature of our extended version of CADICAL to generate proofs both in LRAT and in FRAT, where in FRAT, every step is properly justified. The results show that LRAT-TRIM scales much better than FRAT-RS, although there was a bug which we reported that made FRAT-RS significantly slower when not using the *super strict mode*. Furthermore, LRAT-TRIM can also check proofs directly and it turns out that the additional overhead of this (untrusted) checking compared to parsing and trimming is small.

Overall, our new LRAT proof flow performs best, with reasonably small overhead on solving. To ease visual comparison, we printed all different configurations into a single graph (Figure 1d). The fastest option is (of course) "no-checking" but our new method is not too far behind. Figure 2 shows that the overhead (cost) of proof checking compared to not checking any proofs. Our approach performs best taking only 30% more time than pure solving. The existing competing approaches are much slower with DRAT incurring an overhead of 180% and FRAT still requiring 125% more time than solving, i.e., both more than doubling overall certification time, while our approach has faster checking than solving.

As a sanity check, we also tested our LRAT proof flow using the default shrinking (see Fig 3). We observed that our new approach remains faster compared to the FRAT proof flow, confirming our initial findings.

[1] `https://github.com/digama0/frat/issues/18`

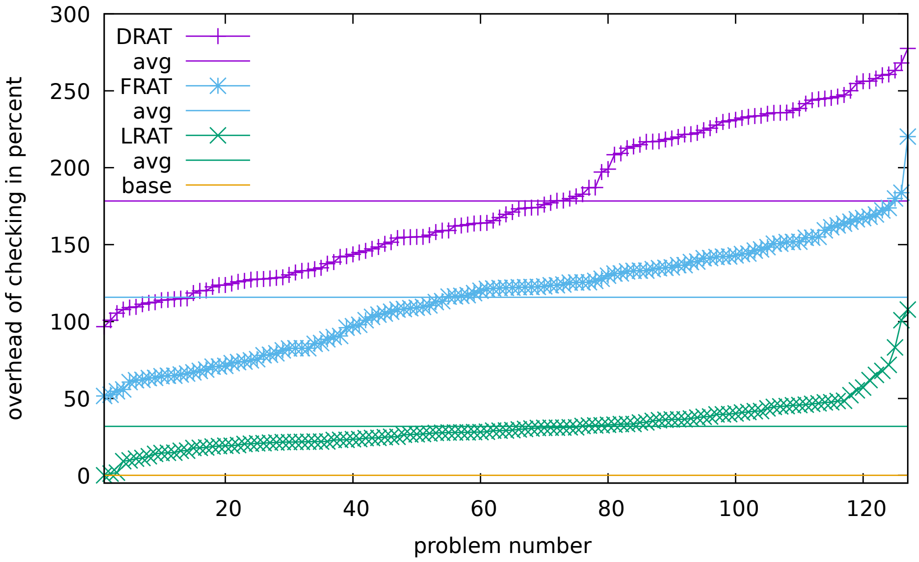


Figure 2 Overhead of the whole checking flow using FRAT, DRAT and our new LRAT flow on top of plain solving (without proof generation and checking), with averages shown as horizontal lines (LRAT 30% overhead, FRAT 125% and DRAT 180% overhead).

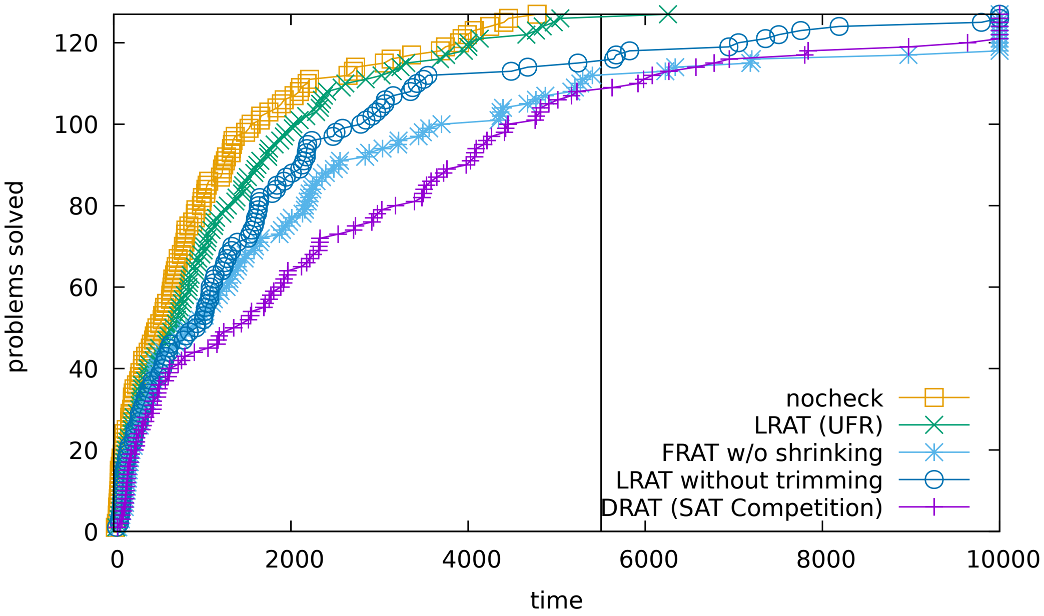


Figure 3 CDF all methods with vertical line indicating timeout of the solver with default options (i.e., with shrinking not supported by the CaDiCaL).

6 Conclusion

We have implemented native LRAT proof production in our SAT solver CaDiCaL. Even though direct production of LRAT proofs slows down the solver slightly this loss is by far offset by the reduction in proof checking time, both compared to DRAT and FRAT proofs. At the end our certification flow adds only 30% overhead compared to pure solving while other approaches take more than twice the time for certification.

It might be interesting to apply this work to recent results on distributed proof generation in the context of the cloud solver Mallob [18] as well as our multi-core solver in Gimsatul [11]. We also see the question of how to handle clause ids for virtual binary clauses as a technical challenge. Such clauses occur in both Gimsatul [11] and the state-of-the-art sequential solver kissat [3].

References

1 Seulkee Baek, Mario Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. *Log. Methods Comput. Sci.*, 18(2), 2022. `doi:10.46298/lmcs-18(2:3)2022`.

2 Armin Biere. Lrat trimmer, Last access, March 2023. Source code. URL: `https://github.com/arminbiere/lrat-trim`.

3 Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, pages 10–11. University of Helsinki, 2022.

4 Armin Biere, Mathias Fleury, and Mathias Heisinger. CaDiCaL, Kissat, Paracooba entering the SAT Competition 2021. In Marijn J. H. Heule, Matti Järvisalo, and Martin Suda, editors, *SAT Competition 2021*, 2021.

5 Armin Biere, Matti Järvisalo, and Bejamin Kiesl. Preprocessing in SAT solving. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 391–435. IOS Press, 2nd edition edition, 2021.

6 Luís Cruz-Filipe, Marijn J. H. Heule, Jr. Hunt, Warren A., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26 – 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2017. `doi:10.1007/978-3-319-63046-5_14`.

7 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing.

8 Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

9 Nick Feng and Fahiem Bacchus. Clause size reduction with all-UIP learning. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 2020. `doi:10.1007/978-3-030-51825-7_3`.

10 Mathias Fleury and Armin Biere. Efficient All-UIP learned clause minimization. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021 – 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2021. `doi:10.1007/978-3-030-80223-3_12`.

11 Mathias Fleury and Armin Biere. Scalable proof producing multi-threaded SAT solving with Gimsatul through sharing instead of copying clauses. *CoRR*, abs/2207.13577, 2022. doi:10.48550/arXiv.2207.13577.

12 Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2008, Fort Lauderdale, Florida, USA, January 2-4, 2008*, 2008. URL: http://isaim2008.unl.edu/PAPERS/TechnicalProgram/ISAIM2008_0008_60a1f9b2fd607a61ec9e0feac3f438f8.pdf.

13 Marijn Heule, Matti Järvisalo, and Armin Biere. Revisiting hyper binary resolution. In Carla P. Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*, volume 7874 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 2013. doi:10.1007/978-3-642-38171-3_6.

14 Marijn J. H. Heule, Matti Järvisalo, and Armin Biere. Clause elimination procedures for CNF formulas. 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 357–371. Springer, 2010. doi:10.1007/978-3-642-16242-8_26.

15 Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning – 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012. doi:10.1007/978-3-642-31365-3_28.

16 Peter Lammich. Efficient verified (UN)SAT certificate checking. *J. Autom. Reason.*, 64(3):513–532, 2020. doi:10.1007/s10817-019-09525-z.

17 Chu-Min Li, Fan Xiao, Mao Luo, Felip Manyà, Zhipeng Lü, and Yu Li. Clause vivification by unit propagation in CDCL SAT solvers. *Artif. Intell.*, 279, 2020. doi:10.1016/j.artint.2019.103197.

18 Dawn Michaelson, Dominik Schreiber, Marijn J. Heule Heule, Benjamin Kiesl-Reiter, and Michael W. Whalen. Unsatisfiability proofs for distributed clause-sharing SAT solvers. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2023*, Lecture Notes in Computer Science, 2023. Accepted, to appear.

19 Florian Pollitt, Mathias Fleury, and Armin Biere. Efficient proof checking with lrat in cadical (work in progress). In Armin Biere and Daniel Große, editors, *24th GMM/ITG/GI Workshop on Methods and Description Languages for Modelling and Verification of Circuits and Systems, MBMV 2023, Freiburg, Germany, March 23-23, 2023*, pages 64–67. VDE, 2023. Accepted. URL: https://cca.informatik.uni-freiburg.de/papers/PolittFleuryBiere-MBMV23.pdf.

20 Florian Pollitt, Mathias Fleury, and Armin Biere. Native LRAT in CaDiCaL for faster proof checking, 2023. URL: https://cca.informatik.uni-freiburg.de/lrat.

21 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation redundancy checking in cakeml. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 223–241. Springer, 2021. doi:10.1007/978-3-030-72013-1_12.

22 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

23 Nathan Wetzler, Marijn J. H. Heule, and Jr. Hunt, Warren A. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014 – 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.

24 Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in Boolean satisfiability solver. In Rolf Ernst, editor, *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2001, San Jose, CA, USA, November 4-8, 2001*, pages 279–285. IEEE Computer Society, 2001. doi:10.1109/ICCAD.2001.968634.

Even Shorter Proofs Without New Variables

Adrián Rebola-Pardo
Technische Universität Wien, Austria
Johannes Kepler Universität Linz, Austria

Abstract

Proof formats for SAT solvers have diversified over the last decade, enabling new features such as extended resolution-like capabilities, very general extension-free rules, inclusion of proof hints, and pseudo-boolean reasoning. Interference-based methods have been proven effective, and some theoretical work has been undertaken to better explain their limits and semantics. In this work, we combine the subsumption redundancy notion from [9] and the overwrite logic framework from [42]. Natural generalizations then become apparent, enabling even shorter proofs of the pigeonhole principle (compared to those from [27]) and smaller unsatisfiable core generation.

2012 ACM Subject Classification Hardware → Theorem proving and SAT solving

Keywords and phrases Interference, SAT solving, Unsatisfiability proofs, Unsatisfiable cores

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.22

Funding This work has been supported by the LIT AI Lab funded by the State of Upper Austria, the LogiCS doctoral program W1255-N23 of the Austrian Science Fund (FWF), the Vienna Science and Technology Fund (WWTF) through grants VRG11-005 and ICT15-103, and Microsoft Research through its PhD Scholarship Programme.

Acknowledgements I would like to thank Georg Weissenbacher for the discussions on WSR, as well as the anonymous reviewers who provided very useful comments.

1 Introduction

The impressive recent improvements in SAT solving have come coupled with the need to ascertain their results. While satisfiability results are straightforward to check, unsatisfiability results require massive proofs, sometimes petabytes in size [28, 24]. The search for proof systems that enable both easy proof generation and smaller proofs has yield many achievements [17, 15, 53, 27, 41, 3, 9, 16, 4].

Modern proof systems rely on redundancy properties presenting a phenomenon known as *interference* [31, 23, 42]. Whereas traditional proof systems derive clauses that are implied by the premises, interference-based proof systems merely require introduced clauses to be consistent with them. Interference proofs preserve the existence of a model throughout the proof, rather than models themselves. A somewhat counterintuitive semantics thus arises: introducing a clause in an interference-based proof system does not only depend on the presence of some clauses, but also on the absence of some other clauses [39, 42].

The most general interference-based proof system in the literature is known as DSR [9]. While its predecesor DPR had success in generating short proofs of the pigeonhole formula without introducing new variables [27], DSR did not seem to succeeded in improving this result, despite being intuitively well-suited for it.

In this work, we analyze the semantics of DSR proofs extending previous work on DPR proofs [42]. We find similar results to that article; in particular, satisfiability-preserving DSR proofs can be reinterpreted as more traditional, DAG-shaped, model-preserving proofs over an extension of propositional logic with a *mutation* operator. Crucially, these DAG-shaped proofs remove the whole-formula dependence interference is characterized by, enabling an easier analysis of the necessary conditions for satisfiability-preservation.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 22; pp. 22:1–22:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This analysis hints at a generalization we call *weak substition redundancy* (WSR [ˈwɪzɚ]), which allows shorter, more understandable, easier to generate, faster to check proofs. We demonstrate this by giving an even shorter proof of the pigeonhole formula. We also provide a couple of examples where smaller unsatisfiable cores can be generated during proof checking, and fewer lemmas are required during proof generation.

Interference-based proofs

Much of proof generation and checking is still done in the same way as a couple decades ago, by logging the sequence of *learnt clauses* in CDCL checkers, sometimes together with antecedents, and checking those clauses for simple entailment criteria such as *reverse unit propagation* (RUP) [17, 54]. Other parts of the proof are generated using more advanced deduction techniques; even their infrequent use can dramatically decrease the size of generated proofs [20, 53, 32, 26], overcoming not only technical limitations in proof generation, but also theoretical bounds [18, 51, 52]. Clause deletion information is also recorded in the proof, which is needed to reduce memory footprint in checking [21].

Much research has been invested on finding ever more powerful proof rules [31, 27, 9] that allow to succintly express inprocessing techniques such as Gaussian elimination [48, 47, 38, 10, 16] or symmetry breaking [1, 2, 22]. These proof rules are collectively called *interference-based rules*, since their derivation depends on the whole formula rather than just on the presence of some specific clauses [31, 23, 39, 42]. One of the most general interference techniques is *substitution redundancy* (SR), which allows a version of reasoning without loss of generality [9]; this technique has been recently lifted to pseudo-Boolean reasoning with impressive results [16].

Substitution redundancy and the pigeonhole problem

A previous version of SR, called *propagation redundancy* (PR) [27], was successful in achieving short proofs of the pigeonhole problem, known for having exponential proofs in resolution [18] and polynomial yet cumbersome proofs in extended resolution [11]. The proof from [27] can be understood in terms of reasoning without loss of generality [42]: it assumes that a given pigeon is in a given pigeonhole, for otherwise we could swap pigeons around.

PR does not have a method to swap the values of variables; rather, it can only conditionally set them to true or false. Hence, linearly many reasoning steps are needed to just to achieve the swap. SR, on the other hand, allows variable swaps, so one could expect that the clause expressing the result of this swap would satisfy the SR property. Surprisingly, it does not; in fact, the clause fails to satisfy a requirement that in its PR version was almost trivial.

Interference and logical dependency

Interference-based proofs do not have a “dependence” or “procedence” structure: since the ability to introduce a clause is contingent on the whole formula, no notion of “antecedents” exists for SR and its predecessors. This becomes a problem when computing unsatisfiable cores and trimmed proofs [37]; it also has the potential to harm the performance of proof checkers, since some techniques that allow skipping unnecessary steps during proof checking are based on logical dependence [19].

This also relates to an issue arising when generating proof fragments for inprocessing techniques. Sometimes, a clause C cannot be introduced as SR because some lemmas are needed; the proof generator might know these lemmas and how to derive them. However, because interference depends on the whole formula, introducing the lemmas before C can further constrain the requirements for C to be introduced, demanding yet more lemmas.

Contributions

Previous work showed that the semantics of PR can be expressed in terms of *overwrite logic* [42]. Overwrite logic extends propositional logic with an *overwrite operator*. Within overwrite logic, DPR proofs can be regarded as DAG-shaped, model-preserving proofs; PR introduction can then be shown to behave as reasoning without loss of generality.

In Section 3 we provide an extension to the overwrite logic framework, called *mutation logic*, which elucidates the semantics of DSR proofs. In particular, model-preserving proofs within mutation logic mimicking satisfiability-preserving DSR proofs can be extracted, as shown in Section 3.1. This allows a clearer understanding of the SR redundancy rule, which in turn makes some improvements over SR apparent.

By introducing minor modifications to the definition of SR, in Section 4 we obtain a new, more powerful redundancy rule called *weak substitution redundancy* (WSR). WSR proofs are more succint than DSR proofs, which we demonstrate by providing a shorter proof of the pigeonhole problem using only $O(n^2)$ clause introductions in Section 5.1.

Furthermore, WSR enables finer-grained ways to reason about dependency in interference-based proofs. This can yield shorter proof checking runtimes and smaller trimmed proofs and unsatisfiability cores when SR clauses are used (Section 5.2), as well as easier proof generation techniques by providing clearer separation for interference lemmas (Section 5.3).

2 Preliminaries

Given a *literal* l, we denote its *complement* as $\bar{l}$. We denote *clauses* by juxtaposing its literals within square brackets, i.e. we denote the clause $l_1 \vee l_2 \vee l_3$ as $[l_1 l_2 l_3]$. We similarly denote conjunctions of literals, called *cubes*, as juxtaposed literals within angle brackets, e.g. $\langle l_1 l_2 l_3 \rangle$. Crucially, we only consider clauses and cubes that do not contain complementary literals, as most SAT solvers and proof checkers already make that assumption. Equivalently, we disallow tautological clauses and unsatisfiable cubes. We also define complementation for clauses and cubes, i.e. $\overline{[l_1 \dots l_n]} = \langle \overline{l_1} \dots \overline{l_n} \rangle$ and $\overline{\langle l_1 \dots l_n \rangle} = [\overline{l_1} \dots \overline{l_n}]$. SAT solving typically operates over formulas in *conjunctive normal form* (CNF), which are conjunctions of clauses. Here we regard CNF formulas as finite sets of clauses.

An *atom* is either a literal, or one of the symbols $\top$ or $\bot$ representing the propositional constants true and false. Complementation is extended to atoms with $\overline{\top} = \bot$ and $\overline{\bot} = \top$. We can then define the usual propositional semantics as follows. A *model* I is a total map from atoms to $\{\top, \bot\}$ such that $I(\top) = \top$ and $\overline{I(l)} = I(\bar{l})$ for all atoms l.

We say that I *satisfies* a literal l (written $I \models l$) whenever $I(l) = \top$. This definition is recursively extended in the usual way to clauses (disjunctively), cubes and CNF formulas (conjunctively). Similarly, we use the typical notions of *entailment* (denoted $\models$), logical *equivalence* ($\equiv$) and *satisfiability*. We also say that a logical expression φ *satisfiability-entails* another expression ψ (denoted $\varphi \models_{\text{sat}} \psi$) whenever, if φ is satisfiable, then ψ is satisfiable too. Similarly, φ is *satisfiability-equivalent* to ψ (denoted $\varphi \equiv_{\text{sat}} \psi$) whenever φ and ψ are either both satisfiable or both unsatisfiable (i.e. $\varphi \models_{\text{sat}} \psi$ and $\psi \models_{\text{sat}} \varphi$).

An *atomic substitution* σ is a total map from atoms to atoms satisfying the following constraints:

(i) $\sigma(\top) = \top$.

(ii) $\overline{\sigma(l)} = \sigma(\bar{l})$ for all atoms l.

(iii) $\sigma(l) \neq l$ only for finitely many atoms l.

This definition is essentially equivalent to the substitutions from [9]. The form presented here makes it easier to compose atomic substitutions with other atomic substitutions, i.e. $(\sigma \circ \tau)(l) = \sigma(\tau(l))$, and with models, i.e. $(I \circ \sigma)(l) = I(\sigma(l))$; the latter is a model that satisfies a given logical expression φ iff I satisfies the expression resulting from applying the substitution σ to φ.

Note that atomic substitutions have a finite representation: only finitely many literals are mapped to atoms other than themselves, and giving the mapping for one polarity fixes the mapping for the other polarity. Hence, one can represent a substitution as a set of mappings $\{x_1 \mapsto l_1, \ldots, x_n \mapsto l_n\}$ where the x_i are pairwise distinct variables, the l_i are atoms, and any variable other than the x_i is mapped to itself.

Our restriction that clauses must be non-tautological is somewhat at odds with the concept of substitutions. An atomic substitution σ *trivializes* a clause C if either:

(a) there is a literal $l \in C$ with $\sigma(l) = \top$.

(b) there are two literals $l, k \in C$ with $\sigma(l) = \overline{\sigma(k)}$.

Applying σ to C yields a tautology whenever σ trivializes C, and a (non-tautological) clause otherwise. Then we can define the *reduct* of a clause C or a CNF formula F by an atomic substitution σ as:

$$C\big|_\sigma = [\sigma(l) \mid l \in C \text{ and } \sigma(l) \neq \bot], \quad \text{if } \sigma \text{ does not trivialize } C$$
$$F\big|_\sigma = \big\{C\big|_\sigma \mid C \in F \text{ and } \sigma \text{ does not trivialize C}\big\}$$

▶ **Lemma 1.** *Let C be a clause, F be a CNF formula, and σ be an atomic substitution. The following then hold:*

(i) *σ trivializes C if and only if $I \circ \sigma \models C$ for all models I.*

(ii) *If σ does not trivialize C, then $I \circ \sigma \models C$ if and only if $I \models C\big|_\sigma$ for all models I.*

(iii) *$I \circ \sigma \models F$ if and only if $I \models F\big|_\sigma$ for all models I.*

Proof. Let us first show (ii). First, observe that I satisfies $C\big|_\sigma$ if and only if I satisfies $\sigma(l)$ for some literal $l \in C$. But this is equivalent to $(I \circ \sigma)(l) = \top$ for some $l \in C$, which is precisely $I \circ \sigma \models C$.

We now show (i). The "only if" implication is straightforward from the definition of a trivializing substitution. For the "if" implication, we show that if σ does not trivialize C, then $I \circ \sigma$ falsifies C for some model I. Claim (ii) gives out that any model I falsifying $C\big|_\sigma$, which exists because it is a (non-tautological) clause, has this property.

Claim (iii) then follows easily from claims (i) and (ii). ◀

Note that, for atomic substitutions that only map variables to the constants $\top$ or $\bot$, there exists a correspondence with cubes. In particular, given variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, the cube Q is bijectively associated to the atomic substitution $Q^\star$ where:

$$Q = \langle x_1 \ldots x_n\, \overline{y_1} \ldots \overline{y_m} \rangle \qquad Q^\star = \{x_1 \mapsto \top, \ldots, x_n \mapsto \top, y_1 \mapsto \bot, \ldots, y_m \mapsto \bot\}$$

2.1 Interference-based redundancy notions

Throughout the last decade, several redundancy notions collectively called *interference-based rules* have appeared in the literature [31, 27, 23, 9]. Originating from clause elimination techniques [29, 30, 33], interference can be also used to introduce clauses in the formula; unlike more classical techniques, though, these clauses do not need to be implied by the formula, but rather *consistent* with it. Specifically, given a CNF formula F, introducing a clause C through interference requires that $F \equiv_{\text{sat}} F \cup \{C\}$.

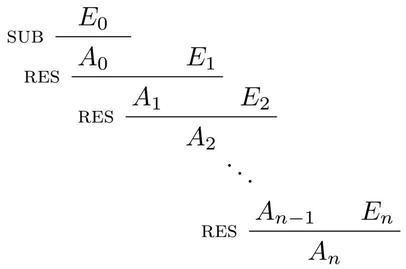


Figure 1 General form of a subsumption-merge chain [39, 43] deriving the clause A_n from premises $E_0, \ldots, E_n$. SUB represents the subsumption rule, so it requires $E_0 \subseteq A_0$. RES represents the resolution rule, which can be applied if there is a literal $l_i \in A_{i-1}$ with $\bar{l}_i \in E_i$; in this case, $A_i = A_{i-1} \setminus \{l_i\} \vee E_i \setminus \{\bar{l}_i\}$. Subsumption-merge chains additionally require that the RES inferences are actually self-subsuming [14], i.e. $E_i \setminus \{\bar{l}_i\} \subseteq A_{i-1}$. Under these conditions, the clause A_n is a RUP clause over any CNF formula containing $E_0, \ldots, E_n$. Conversely, any RUP clause over F can be derived as A_n through a subsumption-merge chain from some clauses $E_0, \ldots, E_n \in F$. In fact, the E_i are the reason clauses used during unit propagation in a RUP check in reverse ordering (up to a topologically-compatible reordering) [15, 39].

Many interference-based rules are based on a criterion for entailment called *reverse unit propagation* (RUP) [17]. A clause C is called a *RUP clause* over a CNF formula F whenever unit propagation applied to F using the assumption literals $\overline{C}$ yields a conflict; under these circumstances, it can be shown that $F \models C$.

RUP clauses can be characterized in terms of resolution proofs. In particular, a clause C is a RUP clause over F if and only if C can be derived from F through a derivation of a particular form, called a *subsumption-merge chain* [39]. These are derivations as shown in Figure 1, starting with a subsumption inference and followed by a number of *resolution merges*, also known as *self-subsuming resolutions* [14]. The specifics of subsumption-merge chains in relation to RUPs are not quite relevant for our discussion; we direct the interested reader to [39, 43]. For us, it suffices to know that checking whether C is a RUP clause over F is essentially the same as finding the subsumption-merge chain that derives C from F [54].

Building on RUP clauses, many redundancy notions can be defined. The most relevant for our discussion are, in increasing generality order, *resolution-asymmetric tautologies* (RATs), *propagation redundancies* (PRs) and *substitution-redundancies* (SRs):

▶ **Definition 2.** *Let C be a clause and F be a CNF formula.*

(i) *We say C is a RAT clause [31] over F upon a literal l whenever $l \in C$ and, for every clause $D \in F$ with $\bar{l} \in D$, the expression $C \vee D \setminus \{\bar{l}\}$ is either a tautology or a RUP clause over F.*

(ii) *We say C is a PR clause [27] over F upon a cube Q whenever $Q \models C$ (i.e. $Q \cap C \neq \emptyset$) and each clause in $F\big|_{Q^\star}$ is a RUP clause over $F\big|_{\overline{C}^\star}$.*

(iii) *We say C is a SR clause [9] over F upon an atomic substitution σ whenever σ trivializes C and each clause in $F\big|_{\sigma}$ is a RUP clause over $F\big|_{\overline{C}^\star}$.*

For a given *witness* (i.e. the literal l, the cube Q or the substitution σ), checking whether a clause C is a RAT/PR/SR clause over F upon the corresponding witness is polynomial over the size of F. In particular, this check takes at most one RUP check for each clause [9]; and RUP checking is quadratic on the size of F [15]. Finding the right witness is nevertheless NP-complete [27]. These redundancy notions satisfy the general condition for interference:

▶ **Theorem 3.** *Let C be a clause and F be a CNF formula, and assume either of the following:*

(a) *C is a RAT clause over F upon a literal l [31].*

(b) *C is a PR clause over F upon some cube Q [27].*

(c) *C is an SR clause over F upon some atomic substitution σ [9].*

Then, $F \equiv_{sat} F \cup \{C\}$.

In this paper we will mostly focus on substitution redundancy, which is the most general of them. However, we will use an equivalent definition, as per [9, Lemma 5]: instead of the condition that each clause in $F\big|_{\sigma}$ is a RUP clause over $F\big|_{\overline{C}^{\star}}$, we require that, for each clause $D \in F$, either σ trivializes D, or $\overline{C} \models D\big|_{\sigma}$, or the clause $C \vee D\big|_{\sigma}$ is a RUP clause over F.

2.2 Proof systems for SAT solving

RUP clauses provided the first effective solution to the problem of certifying an unsatisfiability result from a SAT solver. In particular, learnt clauses in a CDCL SAT solver [45] are RUP clauses [17, 15], so checking that each clause in the list of learnt clauses is a RUP clause over the previously derived formula amounts to certifying that the last clause in the list is entailed by the solved formula. If that clause is the empty clause, the list constitutes a refutation.

However, the proof complexity of RUP proofs is rather poor: there exist many simple problems whose refutations in resolution-based proof systems, such as RUP, are exponential on the size of the refuted formula [18, 51, 52]. In fact, this problem extends to (purely) CDCL SAT solvers, on which these results impose a performance upper bound [40, 5].

To alleviate the impact of these results, some inprocessing techniques were developed, including reencoding of cardinality constraints [6, 35], Gaussian elimination over $\mathbb{Z}_2$ [48, 47] and symmetry breaking [1, 2]. Unfortunately, the aforementioned limitations still apply to the generated refutation, so emitting a RUP proof would still take exponential time.

Allowing interference-based reasoning in the proof led to a vast number of proof formats [20, 53, 27, 13, 12, 34, 49, 9, 4] and proof generation techniques [46, 36, 22, 38, 10, 8, 16, 7]. The proof complexity of these systems is equivalent to that of extended resolution [50, 44, 32, 26], for which no exponential lower bounds are known.

Unlike more traditional, DAG-shaped proofs, interference-based proofs take the form of a list of *clause introductions* and *deletions*. Starting with the input CNF formula F, clause introductions of the form **i:** C add a clause C to F, whereas clause deletions of the form **d:** C remove C from F. At each point in the proof there is an *accumulated formula* where all the previous instructions in the proof have been applied.

Just as DAG-shaped proofs like resolution maintain a soundness invariant (i.e. each model satisfying the premises of the proof also satisfies the conclusion), interference-based proofs are *satisfiability-preserving* [39]: at any point in an interference-based proof of F, the accumulated formula G satisfies $F \models_{\text{sat}} G$. This is guaranteed by imposing some conditions on clause introductions; clause deletions do not have any requirements, because deleting a clause is always satisfiability-preserving.

Different proof systems then arise from different conditions on clause introductions. *Delete Resolution Asymmetric Tautology* (DRAT) requires them to be either RUP clauses or RAT clauses over the accumulated formula [20, 53], and similarly for *Delete Propagation Redundancy* (DPR) [27] and *Delete Substitution Redundancy* (DSR) [9]. Note that, in the case of introducing a RAT/PR/SR clause (as opposed to a RUP clause), the witness ω must be specified; in this case we denote it as **i:** C, ω.

2.3 Overwrite logic

Interference-based proofs represent a structural and semantic departure from traditional proof systems. This is due to the *non-monotonic* properties of SR: an SR clause over F upon σ is not necessarily an SR clause over a formula containing F. [31, 39].

The consequences of non-monotonicity are far-reaching. Interference-based proofs cannot be freely composed as, for example, resolution proofs can [25]: the correctness of a clause introduction depends, in principle, on the whole formula, which motivated the name "interference" as opposed to "inference" [23].

DPR proofs can be seen as model-preserving, tree-shaped, monotonic proofs over a more general logic, known as overwrite logic [42]. There, a model I can be *conditionally overwritten* with an *overwrite rule* of the form $(Q \coloneqq T)$, where Q and T are cubes. Then, the model $I \circ (Q \coloneqq T)$ is defined as $I \circ Q^\star$ if $I \models T$, or as I otherwise. That is, if T is satisfied, then the minimal assignment satisfying Q is overwritten on I. Instead of clauses, overwrite logic deals with *overwrite clauses*, represented as $\nabla \varepsilon_1 \ldots \varepsilon_n . C$, where C is a clause and the $\varepsilon_i = (Q_i \coloneqq T_i)$ are overwrite rules. Such an overwrite clause is satisfied by a model I whenever $I \circ \varepsilon_1 \circ \cdots \circ \varepsilon_n \models C$.

This framework accurately expresses the reasoning performed by PR introduction [42]:

▶ **Theorem 4.** *Let C be a PR clause over a CNF formula F upon a cube Q. Then, the implication $F \models \nabla(Q \coloneqq \overline{C}).(F \cup \{C\})$ holds.*

This result means that non-monotonic, satisfiability-preserving reasoning using PR clauses can be turned into monotonic, model-preserving reasoning in overwrite logic. [42] further introduces a traditional, DAG-shaped proof system over overwrite clauses that mimics PR proofs, hence suggesting that the whole-formula dependence featured by interference-based proof systems can, to some extent, be curbed.

3 Mutation semantics for DSR proofs

The overwrite logic presented in Section 2.3 was designed to formalize the semantics of DPR proofs. In particular, models are overwritten with cubes, which act as witnesses for PR clause introductions. In order to extend this framework to DSR proofs, the role of cubes must now be fulfilled by atomic substitutions. Here we introduce *mutation logic*, which is a straightforward extension of overwrite logic.

In its most general form, a *mutation rule* is an expression $(\sigma \coloneqq \tau)$, where σ is an atomic substitution and τ is any logical expression that can be evaluated under a model. We call τ the *trigger* of the rule, and σ its *effect*. Mutation rules themselves are not logical expressions and they cannot be satisfied or falsified. They are instead intended to codify the idea "if the trigger τ is satisfied, then apply the effect substitution σ". We thus define the *application* of a mutation rule $(\sigma \coloneqq \tau)$ to a model I as:

$$I \circ (\sigma \coloneqq \tau) = \begin{cases} I \circ \sigma & \text{if } I \models \tau \\ I & \text{if } I \not\models \tau \end{cases}$$

As with overwrite logic, the main difference with propositional logic is the inclusion of a mutation operator ∇. As in [42], one can recursively define mutation formulas as either propositional formulas, or expressions of the form $\nabla(\sigma \coloneqq \tau). \varphi$ where σ is an atomic substitution and φ, τ are mutation formulas. The semantics of the mutation operator are given by $I \models \nabla(\sigma \coloneqq \tau). \varphi$ whenever $I \circ (\sigma \coloneqq \tau) \models \varphi$. In other words: evaluating $\nabla(\sigma \coloneqq \tau). \varphi$ corresponds to evaluating a formula φ' obtained from φ by applying the effect σ to φ only if the trigger τ is satisfied.

This framework is very general, but just as discussed in [42], nothing meaningful is lost by introducing some strong restrictions. For the purpose of this paper, we will only consider *cubic* mutation rules of the form $(\sigma := Q)$ where Q is a propositional cube. The logical expressions we will use are of three kinds, where we use $\nabla\vec{\varepsilon}.\,\varphi$ to denote a nested mutation $\nabla\varepsilon_1.\ \dots\nabla\varepsilon_n.\,\varphi$ with cubic mutations ε_i:

- *Mutation clauses* of the form $\nabla\vec{\varepsilon}.\,C$ where C is a propositional clause.
- *Mutation CNF formulas* (MCNF), which are finite sets of mutation clauses. The semantics of MCNF formulas are conjunctive, i.e. they are satisfied if every mutation clause in them is satisfied.
- *Uniformly mutation CNF formulas* (UMCNF) of the form $\nabla\vec{\varepsilon}.\,F$ where F is a propositional CNF formula. ∇ distributes over the propositional connectives, e.g. $\nabla\vec{\varepsilon}.\,(\varphi_1 \wedge \varphi_2) \equiv (\nabla\vec{\varepsilon}.\,\varphi_1) \wedge (\nabla\vec{\varepsilon}.\,\varphi_2)$. Hence, UMCNF can be embedded in the fragment of the MCNF formulas that contain clauses with the same mutation prefix.

Similarly to how overwrite logic allows the expression of PR clauses as model-preserving inferences under an overwrite [42], SR clauses become consequences under a mutation.

▶ **Theorem 5.** *Let F be a CNF formula and C be an SR clause over F upon an atomic mutation σ. Then, $F \models \nabla(\sigma := \overline{C}).\,(F \cup \{C\})$.*

Proof. Let I be any model with $I \models F$. Our goal is to show that the model $I' = I \circ (\sigma := \overline{C})$ satisfies $F \cup \{C\}$. If $I \models C$ holds, then $I' = I$, which satisfies both F and C.

Let us now show the case with $I \not\models C$, where we have $I' = I \circ \sigma$. First observe that, since C is an SR clause upon σ, the clause C is trivialized by σ. Lemma 1 then shows $I' \models C$. Now, consider any clause $D \in F$. By the definition of SR clauses, either σ trivializes D, or $\overline{C} \models D\big|_{\sigma}$, or the clause $C \vee D\big|_{\sigma}$ is a RUP clause over F.

As above, the first case implies $I' \models D$. For the second and third cases, it suffices to show $I \models D\big|_{\sigma}$, since Lemma 1 then proves $I' \models D$. For the second case, this follows from $I \models \overline{C}$. For the third case, it follows from $I \models F$ and $I \not\models C$. We have thus shown that $I' \models F \cup \{C\}$ as we wanted. ◀

As for PR clauses in [42], one can read Theorem 5 as claiming that SR clause introduction (and in general, interference-based reasoning) performs reasoning *without loss of generality.* In particular: C can be assumed in F because, were it not to hold in a given model of F, a transformation, namely the one given by σ, could be applied to the variables such that F is still satisfied after the transformation, and C becomes satisfied too.

3.1 DSR proofs as model-preserving proofs

The entailment in Theorem 5 raises the question whether SR proofs can be equivalently expressed as model-preserving, DAG-shaped proofs over the corresponding mutated clauses. Following [42], we can define a proof system as shown in Figure 2.

▶ **Theorem 6.** *The inference rules in Figure 2 are sound, i.e. any model satisfying the premises of each rule satisfies its conclusion as well.*

Proof. The proofs for RES and SUB are straightforward, since the ∇ operator preserves implications.

For ∇TAUT, consider any model I, and let $I' = I \circ \vec{\varepsilon} \circ (\sigma := \overline{C})$. If $I \circ \vec{\varepsilon} \models C$, then $I' = I \circ \vec{\varepsilon}$, which satisfies C. Otherwise, $I' = I \circ \vec{\varepsilon} \circ \sigma$, and since σ trivializes C we have $I' \models C$.

$$\textsc{res}\ \frac{\nabla\vec{\varepsilon}.\, C \qquad\qquad \nabla\vec{\varepsilon}.\, D}{\nabla\vec{\varepsilon}.\, C\setminus\{l\}\vee D\setminus\{\overline{l}\}}$$

$$\textsc{sub}\ \frac{\nabla\vec{\varepsilon}.\, C}{\nabla\vec{\varepsilon}.\, D}\quad \text{where } C\subseteq D$$

$$\nabla\textsc{taut}\ \frac{}{\nabla\vec{\varepsilon}.\, \nabla(\sigma \coloneqq \overline{C}).\, C}\quad \text{where } \sigma \text{ trivializes } C$$

$$\nabla\textsc{intro}\ \frac{\nabla\vec{\varepsilon}.\, C \qquad {}^{\star}\nabla\vec{\varepsilon}.\, \overline{Q}\vee C\big|_{\sigma}}{\nabla\vec{\varepsilon}.\, \nabla(\sigma \coloneqq Q).\, C}\quad \text{where } \star \text{ is only needed if } Q\not\models C\big|_{\sigma}$$

$$\nabla\textsc{elim}\ \frac{\nabla\vec{\varepsilon}.\, \nabla(\sigma \coloneqq Q).\, C \qquad \nabla\vec{\varepsilon}.\, \nabla(\sigma \coloneqq Q).\, C\big|_{\sigma}}{\nabla\vec{\varepsilon}.\, C\big|_{\sigma}}\quad \text{where } \sigma \text{ does not trivialize } C$$

Figure 2 A proof system over mutation clauses.

Let us now show ∇ELIM correct. Consider any model I satisfying the premises, and call $I' = I\circ\vec{\varepsilon}\circ(\sigma \coloneqq \overline{C})$, so that $I'\models C$ and $I'\models C\big|_{\sigma}$. If $I\circ\vec{\varepsilon}\models Q$, then $I' = I\circ\vec{\varepsilon}\circ\sigma$ satisfies C; then $I\circ\vec{\varepsilon}$ satisfies $C\big|_{\sigma}$ by Lemma 1. Otherwise, $I' = I\circ\vec{\varepsilon}$ satisfies $C\big|_{\sigma}$.

Finally, for ∇INTRO, let I be any model satisfying the premises, and $I' = I\circ\vec{\varepsilon}\circ(\sigma \coloneqq Q)$. If $I\circ\vec{\varepsilon}$ satisfies Q then either it also satisfies $\overline{Q}\vee C\big|_{\sigma}$ or $Q\models C\big|_{\sigma}$. Either way, we can conclude $I\circ\vec{\varepsilon}\models C\big|_{\sigma}$, and since in this case we have $I' = I\circ\vec{\varepsilon}\circ\sigma$, Lemma 1 implies that $I'\models C$. The other case is $I\circ\vec{\varepsilon}\not\models Q$, and in this case $I' = I\circ\vec{\varepsilon}$, which satisfies C. ◀

Upon closer inspection of the proof of Theorem 5, the relation between the SR property and satisfiability-preservation becomes clearer. When each clause $D\in F$ is required that either σ trivializes D, or $\overline{C}\models D\big|_{\sigma}$, or the clause $C\vee D\big|_{\sigma}$ is a RUP clause over F, these conditions enable deriving $\nabla(\sigma \coloneqq \overline{C}).\, D$ through rules ∇TAUT or ∇INTRO hold: the left-hand premise in ∇INTRO just means that C has been derived earlier in the SR proof, while the right-hand premise ensures that $C\vee D\big|_{\sigma}$ can be derived (e.g. as a RUP clause).

On the other hand, ∇TAUT guarantees that $\nabla(\sigma \coloneqq \overline{C}).\, C$ can be derived, since the definition of SR clauses *forces* C to be trivialized by σ. Given that ∇ distributes over $\wedge$, these conditions are proving $F\models\nabla(\sigma \coloneqq \overline{C}).\, F$.

Similar to [42], a translation of a DSR proof into a mutation logic proof then works as follows. At each step in the DSR proof, we consider the list of rules $(\sigma_i \coloneqq \overline{C_i})$ corresponding to each SR clause C_i introduced upon σ_i earlier in the proof; this list is potentially empty, e.g. at the start of the proof. Let us denote this list by $\vec{\varepsilon}$. Then, at that point, all clauses D in the accumulated CNF formula have been derived as mutation clauses $\nabla\vec{\varepsilon}.\, D$ in the translation. The translation then proceeds as follows:

1. Deletions in the DSR proof are not translated.
2. A RUP clause C can be derived through a subsumption-merge chain [39]; rules RES and SUB can express a similar derivation of the mutation version of C.
3. For an SR clause C over a CNF formula F upon an atomic substitution σ, we must derive mutation clauses $D_{\nabla} = \nabla\vec{\varepsilon}.\, \nabla(\sigma \coloneqq \overline{C}).\, D$ for each clause $D\in F\cup\{C\}$.
   a. When σ trivializes D, the mutation clause D_{∇} can be derived as an axiom through ∇TAUT. Note that this case includes the case $D = C$ as well; this detail will become relevant in Section 4.1.
   b. When $\overline{C}\models D\big|_{\sigma}$, the mutation clause D_{∇} can be infered from $\nabla\vec{\varepsilon}.\, D$ through ∇INTRO. We know this premise has been previously derived because $D\in F$.

c. When $C \vee D\big|_\sigma$ is a RUP clause over F, a subsumption-merge chain deriving that clause with premises in F exists [15, 39]. Replacing clauses D' with mutation clauses $\nabla\vec{\varepsilon}.\, D'$, resolution inferences with ∇RES and subsumption inferences with ∇SUB in that proof then yields a derivation of $\nabla\vec{\varepsilon}.\, C \vee D\big|_\sigma$ from previously derived mutation clauses. Finally, the rule ∇INTRO derives D_∇.

4. At the end of the proof, the empty clause $[\,]$ is derived in the SR clause, and the translation has derived the mutated clause $\nabla\varepsilon.\, [\,]$. The identity $[\,] = [\,]\big|_\sigma$ for all substitutions σ ensures that ∇ELIM can be iteratively applied to eliminate all mutation operators, so that $[\,]$ is derived in the translation as well.

4 Extending DSR proofs

Understanding DSR proofs as mutation logic proofs opens the door to finer-grained reasoning about interference-based proofs. Crucially, one of the main issues with interference-based proofs is that deriving a clause involves reasoning over the whole currently derived formula. In particular, interference-based proofs can be highly *non-monotonic*: deleting a clause in the current formula can enable new SR introductions; and conversely, introducing a clause can disable previously available SR introductions.

This is, at first sight, at odds with the translation described in Section 3.1: the proofs we obtain there are model-preserving, DAG-shaped proofs with clear dependencies with other derived clauses. What can be derived in a subproof is never affected by independent proof sub-DAGs, so clause introduction never disables SR introductions. Deletions are even more intriguing, since they do not even exist in the mutation logic framework (just as there is no notion of deletion in a resolution proof DAG).

Another noticeable feature is how differently an SR clause C over F is treated in the definition compared to the clauses $D \in F$. Even if at first sight it might look reasonable to consider different conditions on the premises and on the conclusion, the translation from Section 3.1 uses the same set of inference rules to derive both C_∇ and D_∇.

4.1 Weak substitution redundancy

In the translation, the conditions of the definition are used to guarantee that C_∇ can be derived through a ∇TAUT inference. However, we have *three* rules that can derive this mutated clause, and the three are involved in deriving D_∇ for each $D \in F$. We can thus relax the conditions over C by demanding just the same as for each D: either σ must trivialize C, or $\overline{C} \models C\big|_\sigma$, or the clause $C \vee C\big|_\sigma$ must be a RUP clause over F.

Furthermore, there is nothing in the translation forcing us to derive D_∇ for *each and all* clauses $D \in F$. Rather, we must only do so for those clauses that the proof uses later on. However, even if we do not need D after the SR introduction, we still can use D for the RUP checks of *other* clauses in F. Note that this is not quite the same as deleting D before the SR introduction: doing so could make the RUP checks of other clauses in F fail.

These two details suggest an extension of substitution redundancy, which we call *weak substitution redundancy* (WSR).

▶ **Definition 7.** *A clause C is a WSR clause over a CNF formula F upon an atomic substitution σ modulo a subformula $\Delta \subseteq F$ whenever, for each clause $D \in (F \setminus \Delta) \cup \{C\}$, either of the following holds:*

(a) *σ trivializes D.*

(b) $\overline{C} \models D\big|_\sigma$.

(c) *$C \vee D\big|_\sigma$ is a RUP clause over F.*

▶ **Theorem 8.** *Let C be a WSR clause over a CNF formula F upon an atomic substitution σ modulo a subformula $\Delta \subseteq F$. Then, $F \models \nabla(\sigma := \overline{C}).\,((F \setminus \Delta) \cup \{C\})$ holds. In particular, if F is satisfiable, then so is $(F \setminus \Delta) \cup \{C\}$.*

Proof. Similar to the proof of Theorem 5. The main difference is that $F \models \nabla(\sigma := \overline{C}).\,C$ must now be shown using the same reasoning as $F \models \nabla(\sigma := \overline{C}).\,D$ for $D \in F$. ◀

The complexity of checking a WSR clause introduction is similar to that of a PR/SR check. On the one hand, one extra RUP check might be needed if $C \vee C\big|_\sigma$ is not a tautology; on the other hand, one RUP check is spared for each clause in Δ.

A minor benefit of WSR clauses is that, while not every RUP is a RAT, PR or SR clause, every RUP clause is a WSR clause upon the identity atomic substitution. The reason for this is that the condition that the atomic substitution σ must trivialize $[\,]$ always fails. This allows reasoning about WSR proofs without the need for case discussion.

▶ **Corollary 9.** *Let C be clause and F be a CNF formula. Then, C is a RUP clause over F if and only if C is a WSR clause over F upon the identity atomic substitution modulo $\emptyset$.*

This, together with the embedded notion of deletions as Δ, enables the definition of a proof system with only one rule $\mathbf{w}$: C, $\sigma \setminus \Delta$. This rule introduces clause C and deletes clauses in Δ, and is correct whenever C is a WSR clause over F upon σ modulo Δ. We call this proof system the *WSR proof system.*

5 Applications of WSR proofs

So far, we have not yet shown any benefit of WSR over SR (or that they are not equivalent, for that matter). In this section, we demonstrate techniques using WSR proofs that are unavailable in previously existing interference-based proof systems.

5.1 A shorter proof of the pigeonhole problem

One of the first propositional problems that was found to only have exponential resolution proofs was the pigeonhole problem [18]. While polynomial proofs in the extended resolution system had already been known for a decade [11], these proofs needed to introduce fresh variables to support definitions. However, the seminal work on PR clauses presented a shorter DPR proof that did not use extra variables, using $O(n^3)$ instructions [27].

In [42] an analysis of this proof from the overwrite logic perspective was presented; let us briefly reproduce it here. The pigeonhole problem encodes the unsatisfiable problem "find an assignment of n pigeons to $n-1$ pigeonholes such that no two pigeons share the same hole". We consider variables p_{ij} encoding "pigeon i is in hole j". Let us define the following clauses:

$$\begin{aligned}
H_{in} &= [p_{ij} \mid 1 \le j < n] && \text{for } n > 0 \text{ and } 1 \le i \le n\\
P_{ijk} &= [\overline{p_{ik}}\,\overline{p_{jk}}] && \text{for } 1 \le i < j \text{ and } 1 \le k\\
L_{ijn} &= [\overline{p_{i(n-1)}}\,\overline{p_{nj}}] && \text{for } n > 1,\ 1 \le i < n \text{ and } 1 \le j < n-1\\
R_{in} &= [\overline{p_{i(n-1)}}] && \text{for } n > 1 \text{ and } i < n
\end{aligned}$$

Briefly, H_{in} says that pigeon i stays in some hole $1 \le j < n$; P_{ijk} prevents that pigeons i and j both occupy hole k; L_{ijn} can be read as "if the pigeon i is in the last hole, then hole j does not contain the last pigeon"; and finally R_{in} prevents that pigeon i is in the last hole. The pigeonhole problem for n pigeons is then encoded by

$$\Pi_n = \{H_{in} \mid 1 \le i \le n\} \cup \{P_{ijk} \mid 1 \le i < j \le n \text{ and } 1 \le k < n\}$$

Intuitively, a refutation of Π_n proceeds by noting that, without loss of generality, each pigeon $i < n$ is not in hole $n-1$; were this not the case, one can swap pigeon i with pigeon n (which is not in hole n because that would violate $P_{in(n-1)}$). Then, pigeons $1, \dots, (n-1)$ and holes $1, \dots, (n-2)$ are in the conditions of the pigeonhole problem Π_{n-1}. This process can be iterated until Π_1 is reached, which is trivially unsatisfiable.

The proof from [27] follows this reasoning, but a single PR clause is not expressive enough to encode swaps: the only mutations that it can handle are setting variables to true or false. Thus, the proof first derives clauses L_{ijn} for $1 \le i < n$ and $1 \le j < n-1$ as PR clauses with the cube $Q_{ijn} = \langle \overline{p_{i(n-1)}}\,\overline{p_{nj}} p_{ij} p_{n(n-1)} \rangle$. This encodes the following reasoning: without loss of generality, if the pigeon i is in the last hole, then hole j does not contain the last pigeon; were this not the case, ensure that pigeon i is not in the last hole but in the hole j instead, and that the last pigeon is not in hole j but in the last hole instead. Once the clauses L_{ijn} have been derived for each $1 \le i < n$, the clause R_{in} ensuring that pigeon i is not in the last hole can be derived as a RUP clause.

When considered together, the mutations $Q^{\star}_{ijn}$ for $1 \le j < n-1$ express the atomic substitution that swaps pigeons i and n, that is:

$$\sigma_{in} = \{p_{ij} \mapsto p_{nj}, p_{nj} \mapsto p_{ij} \mid 1 \le j < n\} \quad \text{for } 1 \le i < n$$

DSR can handle this kind of mutation. Let us write a DSR derivation of Π_{n-1} from Π_n (where we are omitting some trailing deletions for simplicity):

$$(\mathbf{i{:}}\ R_{1n},\ \sigma_{1n}), \dots, (\mathbf{i{:}}\ R_{(n-1)n},\ \sigma_{(n-1)n}), (\mathbf{i{:}}\ H_{1(n-1)}), \dots, (\mathbf{i{:}}\ H_{(n-1)(n-1)})$$

Clauses $H_{i(n-1)}$ can be introduced as RUP clauses, since they result from resolution on H_{in} and R_{in}. Furthermore, one would hope for the R_{in} clauses to be SR clauses over the preceding formula upon σ_{in}. Let us check this. For each clause D in the preceding formula F, we need to check that either of the following holds:

(a) D is trivialized by σ_{in}
(b) $\langle p_{i(n-1)} \rangle \models D|_{\sigma_{in}}$
(c) the clause $D_{\nabla} = [\overline{p_{i(n-1)}}] \vee D|_{\sigma_{in}}$ is a RUP clause over F.

Checking case by case one can see that the reduct $D|_{\sigma_{in}}$ is always another clause in F, so D_{∇} is either a tautology or can be derived by subsumption from F (which implies it is a RUP clause).

The clause R_{in}, nevertheless, is not an SR clause over F upon σ_{in}, because it is not trivialized by σ_{in}. Observe, however, that

$$C_{\nabla} = C \vee C|_{\sigma_{in}} = [\overline{p_{i(n-1)}}\,\overline{p_{n(n-1)}}] = P_{in(n-1)} \in F$$

In particular, C_{∇} it is a RUP clause over F. Hence, R_{in} is in fact a WSR clause over F upon σ_{in} modulo $\emptyset$. Hence, we can define the WSR derivation π_n of Π_1 from Π_n given in Figure 3 for $n > 1$. The derivation π_n has $O(n^2)$ instructions, and is in fact a refutation, since $[] \in \Pi_1$.

5.2 Smaller cores and shorter checking runtime

SAT solvers generate proofs which often introduce clauses uninvolved in the derivation of a contradiction. This is practically unavoidable because of how solvers generate proofs: solvers mostly just log every learnt clause [17], and at that point the solver does not know what learnt clauses will be useful.

State-of-the-art proof checkers thus validate the proof backwards [19, 53]. Starting from the empty clause at the end of the proof, the checker finds out what clauses are needed to derive each clause as a RUP clause. Required clauses are then *marked*; as the checker

Figure 3 A WSR refutation π_n of the pigeonhole problem Π_n for $n \geq 1$ containing only $O(n^2)$ instructions. Here, π_1 is the empty list and id represents the identity atomic substitution.

$$
\begin{array}{l}
\textbf{w:}\ R_{1n},\ \sigma_{1n} \setminus \emptyset \\
\textbf{w:}\ R_{2n},\ \sigma_{2n} \setminus \emptyset \\
\qquad \vdots \\
\textbf{w:}\ R_{(n-1)n},\ \sigma_{(n-1)n} \setminus \emptyset \\
\textbf{w:}\ H_{1(n-1)},\ \mathrm{id} \setminus \{R_{1n}\} \\
\textbf{w:}\ H_{2(n-1)},\ \mathrm{id} \setminus \{R_{2n}\} \\
\qquad \vdots \\
\textbf{w:}\ H_{(n-1)(n-1)},\ \mathrm{id} \setminus (\{R_{1n}\} \cup (\Pi_n \setminus \Pi_{n-1})) \\
\pi_{n-1}
\end{array}
$$

proceeds backwards, unmarked clauses are skipped. If one were to visualize a RUP proof as a DAG, this amounts to only checking the connected component that actually derives the empty clause while disregarding all other connected components in the DAG.

Backwards checking has three interesting consequences. First, it vastly improves checking runtime: not only are checks for unmarked clauses skipped, but also their premises are skipped as well (unless they are used to derive another marked clause). Second, a shorter, *trimmed* proof can be extracted as a by-product of checking. Finally, by the time the checker reaches the start of the proof, the marked clauses in the input formula form a (not necessarily minimal) unsatisfiable core.

Backwards checking in interference-based proofs

Interference-based proofs do not have DAG-like dependencies as RUP proofs have. Let us formalize the problem of backwards checking in this situation. We assume that the checker keeps track of a CNF formula F and marked clauses $M \subseteq F$ as it proceeds backwards through the proof. When a RAT/PR/SR introduction **i:** C, ω is reached with $C \in M$, the checker removes C from both the formula F and the marked clauses M and validates the corresponding RAT/PR/SR introduction. The goal then is to find some (preferably small) subformula M' with $M \subseteq M' \subseteq F$ such that C is a RAT/PR/SR clause upon ω over M'; this will be the new set of marked clauses.

In the best case scenario, C satisfies the corresponding redundancy property over M, so the checker can move on with $M' = M$. There is only one way the redundancy property might not hold over M: when one of the RUP checks from Definition 2 fails over M (but still succeeds over F), the premises of the induced subsumption-merge chain must become marked; let us (conspicuously) call this set of *newly* marked clauses Δ. The problem we are tackling is whether clauses in Δ *really* need their own RUP check as mandated by Definition 2.

For RAT, it turns out, they do not: one can show that, for a witness literal l in a RAT check, the clauses in Δ never contain $\bar{l}$, so they never trigger further RUP checks. Such a convenient coincidence does not hold for PR or SR, though. In order to establish that C is a PR/SR clause upon some M', the clauses in Δ must undergo their own RUP check, which might add new clauses to Δ, and so on until fixpoint.

This is nevertheless wasteful. By the time the first Δ has been computed, introducing C can already be claimed to be satisfiability-preserving, *just not as a PR/SR*: the conditions above prove that C is a WSR clause upon ω over $M \cup \Delta$ modulo Δ. This means that a proof checker (even one that only checks PR/SR) can simply set $M' = M \cup \Delta$ and continue checking the rest of the proof.

To the best of our knowledge, existing checkers do not deal with this situation in an optimal way, e.g. the reference DPR checker `dpr-trim` resorts instead to the fixpoint method[1]. Note that the fixpoint method always produces a larger M' than the WSR-based method, with associated longer runtimes, larger unsatisfiability cores and longer trimmed proofs.

Even if for (uncertified) checking WSR only seems relevant at a theoretical level, state-of-the-art proof checkers emit trimmed, annotated proofs that can be further checked with a verified tool [12, 49]. The formats these annotated proofs use, such as LRAT or LPR, are based on RAT/PR, and so the fixpoint method is needed if an annotated proof must be emitted in one of these formats. Either way, the need for the fixpoint method could be removed by emitting WSR-based annotated proofs.

▶ **Example 10.** Let us define three CNF formulas. The formula M contains clauses:

$$[a\,\overline{c}\,x] \qquad [\overline{a}\,\overline{u}\,\overline{v}\,\overline{x}] \qquad [c\,\overline{u}\,\overline{v}\,x] \qquad [a\,\overline{x}\,\overline{y}\,\overline{z}] \qquad [a\,\overline{c}\,\overline{x}\,y] \qquad [\overline{a}\,b\,u]$$
$$[c\,u] \qquad [\overline{u}\,y\,z] \qquad [\overline{a}\,\overline{b}\,\overline{c}] \qquad [c\,\overline{x}\,\overline{z}]^{\bullet} \qquad [\overline{c}\,\overline{x}z]^{\bullet} \qquad [c\,\overline{x}\,\overline{y}]^{\bullet} \qquad [\overline{a}\,b\,\overline{u}\,v\,\overline{x}]^{\bullet}$$

The formula Δ contains:

$$[b\,\overline{u}\,x] \qquad [\overline{b}\,\overline{t}\,v\,x\,\overline{y}] \qquad [\overline{b}\,t\,v\,x\,\overline{z}] \qquad [t\,v\,\overline{y}\,z] \qquad [\overline{t}\,v\,y\,\overline{z}]$$

Finally, $\Gamma = \left\{ \left[\overline{b}\,x\,\overline{u}\,y\,\overline{z}\right] \right\}$. Let us assume that a proof checker is checking a DSR refutation of the unsafistiable formula $F = M \cup \Delta \cup \Gamma$ backwards. It eventually reaches the first instruction, an SR clause introduction for $C = [x\,\overline{u}]$ upon the atomic substitution $\sigma = \{x \mapsto \top, a \mapsto \top, v \mapsto \bot, t \mapsto \top\}$. At this point, the clauses in M (in addition to C) have been marked for checking; since this is the first instruction, the marked clauses after checking C for SR are an unsatisfiable core of F. One can check that σ trivializes C, and that all clauses in M except for the ones highlighted with • satisfy the conditions in Definition 2 using propagation clauses exclusively from M. For the highlighted clauses, propagating with clauses from $M \cup \Delta$ does suffice to satisfy Definition 2.

As we learnt in Section 4, we can now stop checking: C is a WSR clause over the formula $M \cup \Delta$ modulo Δ; the newly marked clauses (which form the generated unsatisfiable core) are thus $M \cup \Delta$. Current checkers will nevertheless not stop here, since SR is more restrictive than WSR. In particular, they check the newly marked clauses Δ for the conditions in Definition 2 as well. As it turns out, C is not even an SR clause over $M \cup \Delta$, but only over F: for the RUP check for $C \vee \left[\overline{t}\,v\,y\,\overline{z}\right]\big|_{\sigma}$ to succeed, the clause in Γ is needed too. That clause becomes subsequently marked, and a further check is performed for it. This check finally succeeds, reaching a fixpoint.

This example shows that SR marks strictly more clauses than WSR, which translates into larger generated unsatisfiable cores and trimmed proofs, as well as a longer checking runtime since the extra marked clauses will be themselves checked.

5.3 Interference-free interference lemmas

The differences between SR and WSR presented in Section 5.2 can too be exploited during proof generation. While the largest share of a proof generated by a state-of-the-art SAT solver consists of learnt clauses introduced as RUP clauses as well as clause deletions, inprocessing techniques also contribute to the proof. Typically, an inprocessing technique performs some

[1] See `https://github.com/marijnheule/dpr-trim/blob/83eb40b9028100aca63a419eb6d08b45acf517ad/dpr-trim.c`, line 660.

reasoning and then a (to some extent) hardcoded proof fragment of the results is generated. No proof search is performed; rather, the specialized reasoning performed by the inprocessing technique is translated into the target proof system by a method that has previously been proven correct (on paper, not *in silico*).

Interference-based proof systems are notable for their ability to generate succint proof fragments for many inprocessing techniques and non-CDCL methods, including parity reasoning [38, 16], symmetry breaking [22] and BDD-based reasoning [7]. Devising these proofs is complex for several reasons; among them is that, in an interference-based proof system, introduced lemmas may need further lemmas for satisfy Definition 2.

Let us assume we want to generate a proof fragment deriving a clause C as an SR clause from F upon some atomic substitution σ. The clause C has been obtained through some inprocessing technique, and we know that all the clauses $D_\nabla = C \vee D\big|_\sigma$ for $D \in F$ are implied by C because of some property of the inprocessing technique. However, we might find that some of the D_∇ are not RUPs over F; after all, RUP is just a criterion for entailment. We can derive some additional clauses (i.e. lemmas) $L^1, \dots, L^n$ from F such that D_∇ is a RUP over $F \cup L^1, \dots, L^n$, but now the definition of SR clauses demands that the L^i_∇ are RUP clauses as well, which might need additional clauses and so on.

This is, in essence, the proof generation version of the proof checking situation from Section 5.2. Just as we did there, with WSR we can completely bypass the need to prove that the L^i_∇ are RUP clauses: C is already a WSR clause over $F \cup \left\{L^1, \dots, L^n\right\}$ upon σ modulo $\left\{L^1, \dots, L^n\right\}$. In other words, WSR allows introducing interference lemmas that need not be taken into account for RUP checks.

▶ **Example 11.** Let us consider the CNF formula F containing clauses:

$$[a\,b\,\overline{x}\,y] \qquad [a\,b\,x\,y\,\overline{z}] \qquad [a\,b\,x\,z] \qquad [\overline{a}\,u\,v] \qquad [\overline{c}\,u\,v] \qquad \left[a\,c\,\overline{b}\,y\right] \qquad [a\,c\,b\,\overline{y}]$$
$$\left[c\,\overline{b}\,\overline{y}\,\overline{z}\right] \qquad \left[c\,\overline{b}\,x\,\overline{y}\,z\right] \qquad \left[c\,\overline{b}\,\overline{x}\,z\right] \qquad [\overline{u}\,v] \qquad [u\,\overline{v}] \qquad [\overline{u}\,\overline{v}]$$

We want to derive the clause $C = [x]$. Unfortunately, C is not a RUP clause over F, so we try to introduce it as an SR clause upon the atomic substitution $\sigma = \{x \mapsto \top, y \mapsto z, z \mapsto y\}$. This *almost* works: all the conditions in Definition 2 hold, except for $C \vee \left[\overline{b}\,c\,\overline{x}\,z\right]\big|_\sigma = \left[\overline{b}\,c\,x\,y\right]$ not being a RUP clause over F. We can derive some RUP lemmas from F, for example $L_1 = [\overline{a}\,y\,v]$ and then $L_2 = [\overline{a}\,y]$; the clause $\left[\overline{b}\,c\,x\,y\right]$ is indeed a RUP clause over $F \cup \{L_1, L_2\}$.

Here is where WSR and SR show their differences again. Under WSR, we can already introduce C in F, because the paragraph above implies that C is a WSR clause over $F \cup \{L_1, L_2\}$ upon σ modulo $\{L_1, L_2\}$. This is not the case for SR, though: because the clause $C \vee L_2\big|_\sigma = [\overline{a}\,x\,z]$ is not a RUP clause over $F \cup \{L_1, L_2\}$, the clause C is not SR over F upon σ. We would need to find additional lemmas to make it so, which might then need further lemmas themselves.

6 Conclusion

We have presented a generalization of the SR redundancy notion, called weak substitution redundancy (WSR). This extension is straightforward once the semantics of interference have been understood, which we achieve by extending the overwrite logic framework from [42] into mutation logic, which is able to handle atomic substitutions.

The main differences between SR and WSR are the weakening of one unnecessarily strong condition in the definition, and the specification of a set of clauses that can be used for ensuring the interference conditions but will not participate in interference themselves.

These minor differences have an impact on the versatility of the proof system. Shorter proofs can be obtained, lemmas can be used in a less obstrusive way, the efficiency of the backwards checking algorithm is enhanced, and smaller unsatisfiable cores and trimmed proofs can be generated.

References

1 Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Solving difficult instances of boolean satisfiability in the presence of symmetry. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 22(9):1117–1137, 2003. doi:10.1109/TCAD.2003.816218.

2 Fadi A. Aloul, Karem A. Sakallah, and Igor L. Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Trans. Computers*, 55(5):549–558, 2006. doi:10.1109/TC.2006.75.

3 Johannes Altmanninger and Adrián Rebola-Pardo. Frying the egg, roasting the chicken: unit deletions in DRAT proofs. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 61–70. ACM, 2020. doi:10.1145/3372885.3373821.

4 Seulkee Baek, Mario M. Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 59–75. Springer, 2021. doi:10.1007/978-3-030-72016-2_4.

5 Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res.*, 22:319–351, 2004. doi:10.1613/jair.1410.

6 Armin Biere, Daniel Le Berre, Emmanuel Lonca, and Norbert Manthey. Detecting cardinality constraints in CNF. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014 – 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 285–301. Springer, 2014. doi:10.1007/978-3-319-09284-3_22.

7 Randal E. Bryant, Armin Biere, and Marijn J. H. Heule. Clausal proofs for pseudo-boolean reasoning. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2022. doi:10.1007/978-3-030-99524-9_25.

8 Randal E. Bryant and Marijn J. H. Heule. Generating extended resolution proofs with a bdd-based SAT solver. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 2021. doi:10.1007/978-3-030-72016-2_5.

9 Sam Buss and Neil Thapen. DRAT proofs, propagation redundancy, and extended resolution. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019 – 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2019. doi:10.1007/978-3-030-24258-9_5.

10 Leroy Chew and Marijn J. H. Heule. Sorting parity encodings by reusing variables. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2020. doi:10.1007/978-3-030-51825-7_1.

11 Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, 1976. doi:10.1145/1008335.1008338.

12 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26 – 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2017. doi:10.1007/978-3-319-63046-5_14.

13 Luís Cruz-Filipe, João Marques-Silva, and Peter Schneider-Kamp. Efficient certified resolution proof checking. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 118–135, 2017. doi:10.1007/978-3-662-54577-5_7.

14 Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005. doi:10.1007/11499107_5.

15 Allen Van Gelder. Producing and verifying extremely large propositional refutations – have your cake and eat it too. *Ann. Math. Artif. Intell.*, 65(4):329–372, 2012. doi:10.1007/s10472-012-9322-x.

16 Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-boolean proofs. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3768–3777. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16494.

17 Evguenii I. Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10886–10891. IEEE Computer Society, 2003. doi:10.1109/DATE.2003.10008.

18 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

19 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 181–188. IEEE, 2013. URL: http://ieeexplore.ieee.org/document/6679408/.

20 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24 – 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2013. doi:10.1007/978-3-642-38574-2_24.

21 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Bridging the gap between easy generation and efficient verification of unsatisfiability proofs. *Softw. Test. Verification Reliab.*, 24(8):593–607, 2014. doi:10.1002/stvr.1549.

22 Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Expressing symmetry breaking in DRAT proofs. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction – CADE-25 – 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 591–606. Springer, 2015. doi:10.1007/978-3-319-21401-6_40.

23 Marijn Heule and Benjamin Kiesl. The potential of interference-based proof systems. In Giles Reger and Dmitriy Traytel, editors, *ARCADE 2017, 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements, Gothenburg, Sweden, 6th August 2017*, EPiC Series in Computing, pages 51–54. EasyChair, 2017. URL: https://easychair.org/publications/paper/TWVW, doi:10.29007/vr7n.

24 Marijn J. H. Heule. Schur number five. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6598–6606. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16952.

25 Marijn J. H. Heule and Armin Biere. Compositional propositional proofs. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning – 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2015. doi:10.1007/978-3-662-48899-7_31.

26 Marijn J. H. Heule and Armin Biere. What a difference a variable makes. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*, volume 10806 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2018. doi:10.1007/978-3-319-89963-3_5.

27 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In Leonardo de Moura, editor, *Automated Deduction – CADE 26 – 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2017. doi:10.1007/978-3-319-63046-5_9.

28 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016. doi:10.1007/978-3-319-40970-2_15.

29 Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2010. doi:10.1007/978-3-642-12002-2_10.

30 Matti Järvisalo, Armin Biere, and Marijn Heule. Simulating circuit-level simplifications on CNF. *J. Autom. Reason.*, 49(4):583–619, 2012. doi:10.1007/s10817-011-9239-9.

31 Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning – 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012. doi:10.1007/978-3-642-31365-3_28.

32 Benjamin Kiesl, Adrián Rebola-Pardo, and Marijn J. H. Heule. Extended resolution simulates DRAT. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning – 9th International Joint Conference, IJCAR 2018, Held as Part*

of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, volume 10900 of *Lecture Notes in Computer Science*, pages 516–531. Springer, 2018. doi:10.1007/978-3-319-94205-6_34.

33 Benjamin Kiesl, Martina Seidl, Hans Tompits, and Armin Biere. Local redundancy in SAT: generalizations of blocked clauses. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:3)2018.

34 Peter Lammich. Efficient verified (UN)SAT certificate checking. In Leonardo de Moura, editor, *Automated Deduction – CADE 26 – 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 237–254. Springer, 2017. doi:10.1007/978-3-319-63046-5_15.

35 Norbert Manthey, Marijn Heule, and Armin Biere. Automated reencoding of boolean formulas. In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Hardware and Software: Verification and Testing – 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2012. doi:10.1007/978-3-642-39611-3_14.

36 Norbert Manthey and Tobias Philipp. Formula simplifications as DRAT derivations. In Carsten Lutz and Michael Thielscher, editors, *KI 2014: Advances in Artificial Intelligence – 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings*, volume 8736 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014. doi:10.1007/978-3-319-11206-0_12.

37 Alexander Nadel, Vadim Ryvchin, and Ofer Strichman. Efficient MUS extraction with resolution. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 197–200. IEEE, 2013. URL: http://ieeexplore.ieee.org/document/6679410/.

38 Tobias Philipp and Adrián Rebola-Pardo. DRAT proofs for XOR reasoning. In Loizos Michael and Antonis C. Kakas, editors, *Logics in Artificial Intelligence – 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings*, volume 10021 of *Lecture Notes in Computer Science*, pages 415–429, 2016. doi:10.1007/978-3-319-48758-8_27.

39 Tobias Philipp and Adrián Rebola-Pardo. Towards a semantics of unsatisfiability proofs with inprocessing. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 65–84. EasyChair, 2017. URL: https://easychair.org/publications/paper/V8G, doi:10.29007/7jgq.

40 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. doi:10.1016/j.artint.2010.10.002.

41 Adrián Rebola-Pardo and Luís Cruz-Filipe. Complete and efficient DRAT proof checking. In Nikolaj Bjørner and Arie Gurfinkel, editors, *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 – November 2, 2018*, pages 1–9. IEEE, 2018. doi:10.23919/FMCAD.2018.8602993.

42 Adrián Rebola-Pardo and Martin Suda. A theory of satisfiability-preserving proofs in SAT solving. In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 583–603. EasyChair, 2018. URL: https://easychair.org/publications/paper/zr7z, doi:10.29007/tc7q.

43 Adrián Rebola-Pardo and Georg Weissenbacher. RAT elimination. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 423–448. EasyChair, 2020. URL: https://easychair.org/publications/paper/cMtF, doi:10.29007/fccb.

44 Robert A. Reckhow. *On the lengths of proofs in the propositional calculus.* PhD thesis, University of Toronto, 1975.

45 João P. Marques Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, pages 220–227. IEEE Computer Society / ACM, 1996. doi:10.1109/ICCAD.1996.569607.

46 Carsten Sinz and Armin Biere. Extended resolution proofs for conjoining bdds. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *Computer Science – Theory and Applications, First International Computer Science Symposium in Russia, CSR 2006, St. Petersburg, Russia, June 8-12, 2006, Proceedings*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006. doi:10.1007/11753728_60.

47 Mate Soos. Enhanced gaussian elimination in dpll-based SAT solvers. In Daniel Le Berre, editor, *POS-10. Pragmatics of SAT, Edinburgh, UK, July 10, 2010*, volume 8 of *EPiC Series in Computing*, pages 2–14. EasyChair, 2010. URL: https://easychair.org/publications/paper/j1D, doi:10.29007/g7ss.

48 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing – SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 – July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. doi:10.1007/978-3-642-02777-2_24.

49 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation redundancy checking in cakeml. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 – April 1, 2021, Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 223–241. Springer, 2021. doi:10.1007/978-3-030-72013-1_12.

50 G. S. Tseitin. On the complexity of derivation in propositional calculus. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer Berlin Heidelberg, 1983. doi:10.1007/978-3-642-81955-1_28.

51 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987. doi:10.1145/7531.8928.

52 Alasdair Urquhart. The symmetry rule in propositional logic. *Discret. Appl. Math.*, 96-97:177–193, 1999. doi:10.1016/S0166-218X(99)00039-6.

53 Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014 – 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.

54 Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 10880–10885. IEEE Computer Society, 2003. doi:10.1109/DATE.2003.10014.

Reducing Acceptance Marks in Emerson-Lei Automata by QBF Solving

Tereza Schwarzová
Masaryk University, Brno, Czech Republic

Jan Strejček
Masaryk University, Brno, Czech Republic

Juraj Major
Masaryk University, Brno, Czech Republic

Abstract

This paper presents a novel application of QBF solving to automata reduction. We focus on *Transition-based Emerson-Lei automata (TELA)*, which is a popular formalism that generalizes many traditional kinds of automata over infinite words including Büchi, co-Büchi, Rabin, Streett, and parity automata. Transitions in a TELA are labelled with acceptance marks and its accepting formula is a positive Boolean combination of atoms saying that a particular mark has to be visited infinitely or finitely often. Algorithms processing these automata are often very sensitive to the number of acceptance marks. We introduce a new technique for reducing the number of acceptance marks in TELA based on satisfiability of *quantified Boolean formulas (QBF)*. We evaluated our reduction technique on TELA produced by state-of-the-art tools of the libraries Owl and Spot and by the tool `ltl3tela`. The technique reduced some acceptance marks in automata produced by all the tools. On automata with more than one acceptance mark obtained by translation of LTL formulas from literature with tools Delag and Rabinizer 4, our technique reduced 27.7% and 39.3% of acceptance marks, respectively. The reduction was even higher on automata from random formulas.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Automata over infinite objects

Keywords and phrases Emerson-Lei automata, TELA, automata reduction, QBF, telatko

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.23

Supplementary Material *Software*: `https://gitlab.fi.muni.cz/xschwar3/telatko`
archived at `swh:1:dir:ce6c69759d7317f0bc9cb2dc2cc96e9473ca31cd`

Funding T. Schwarzová received funding from the European Union's Horizon Europe program under the Grant Agreement No. 101087529.

1 Introduction

Automata over infinite words like Büchi, Rabin, Streett, or parity automata play a crucial role in many algorithms related to concurrency theory, game theory, and formal methods in general. In particular, they are used in specification, verification, analysis, monitoring, and synthesis of various systems with infinite behaviour. In 1987, Emerson and Lei [12] introduced automata over infinite words where acceptance conditions are arbitrary combinations of acceptance primitives saying that a certain set of states should be visited finitely often or infinitely often. In 2015, the same kind of acceptance condition was described in the *Hanoi omega-automata format (HOAF)* [3]. The only difference is that the acceptance primitives talk about finitely or infinitely often visited acceptance marks rather than sets of states. Acceptance marks are placed on transitions and each mark identifies the set of transitions containing this mark. Hence, these automata are called *transition-based Emerson-Lei automata (TELA)* and they generalize many traditional kinds of automata over infinite words including Büchi, co-Büchi, Rabin, Streett, and parity automata.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 23; pp. 23:1–23:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

TELA have attracted a lot of attention during the last few years [5, 15, 16, 18]. Their popularity comes probably from the fact that these automata can often use fewer states than equivalent automata with simpler acceptance conditions. Further, algorithms handling TELA can automatically handle all automata with traditional acceptance conditions. TELA can be obtained for example by translating formulas of *linear temporal logic (LTL)* [17] with tools `ltl2dela` (known as Delag) [16] or `ltl2dgra` (known as Rabinizer 4) [13] of the Owl library, `ltl2tgba` of the Spot library [9], or `ltl3tela` [15]. There are also algorithms processing these automata, for example the emptiness check [5] or translation of TELA to parity automata [18, 7].

Algorithms processing TELA are often sensitive to the number of acceptance marks more than to other parts of the automaton. For example, the transformation of TELA to parity automata based on *color appearance record* [18] transforms a TELA with m acceptance marks and s states into a parity automaton with up to $m! \cdot s$ states. Further, the emptiness-check algorithm [5] is exponential in the number of acceptance marks that appear in acceptance primitives saying that a mark has to be visited finitely often, while it is only polynomial in other measures of the input automaton.

The number of acceptance marks can be algorithmically reduced to one as every TELA can be transformed to an equivalent Büchi automaton (this can be easily done for example by Spot [9]), but this reduction is paid by dramatic changes of state space: the number of states can increase exponentially in the number of acceptance marks and some important structural properties like determinism can be lost. This motivates our study of a technique reducing the number of acceptance marks without altering the structure of the automaton.

Our reduction technique is heavily based on *quantified Boolean formulas (QBF)*. For a given TELA and parameters C, K, it produces a QBF which is satisfiable if and only if there exists an automaton with the same structure, K acceptance marks, an acceptance formula in disjunctive normal form with C cubes (i.e., conjunction of literals), and the same set of accepting runs as the original automaton. The placement of the marks on transitions and the acceptance formula can be obtained from a model of the formula. Besides this formula, we describe also the construction of two simpler formulas whose satisfiability implies the existence of an automaton with the same structure, K acceptance marks, and the same set of accepting runs, but not vice versa.

We have implemented our reduction technique in a tool called `telatko`. We show that the tool can reduce acceptance marks in automata produced by Delag [16], Rabinizer 4 [13] (both included in the Owl library), Spot [9], and `ltl3tela` [15]. While the reduction is relatively modest on TELA produced by `ltl3tela` and Spot, it is substantial on automata produced by the tools of the Owl library.

Related results

There is a simple technique [4] reducing the number of acceptance marks in *transition-based generalised Büchi automata (TGBA)* without changing its structure. We are not aware of any existing research aimed at simplification of acceptance formulas of TELA or reduction of the number of its acceptance marks without increasing the number of states. There exists only a SAT-based approach that transforms a deterministic TELA to an equivalent automaton with a given acceptance condition and a given number of states [2] (if such an automaton exists). Further, there are some SAT-based approaches aimed to reduce the number of states of automata over infinite words. More precisely, there are reductions designed for nondeterministic Büchi automata [11], deterministic Büchi automata [10], and deterministic generalized Büchi automata [1]. Note that these techniques are usually very slow and their authors typically suggest to use them only for specific purposes like looking for cases where some automata construction can be improved.

Casares, Colcombet, and Fijalkow very recently introduced a structure called *alternating cycle decomposition (ACD)* [6] which compactly represents all accepting and non-accepting automata cycles. We expect that ACD could be used to reduce the number of acceptance marks or to simplify the acceptance condition. However, such a reduction is not obvious.

Structure of the paper

The next section introduces basic terms used in the paper. Section 3 explains the construction of the three mentioned quantified Boolean formulas. The reduction algorithm based on these formulas is presented in Section 4. Section 5 describes our tool `telatko` implementing the reduction technique. Experimental results are shown in Section 6. Finally, Section 7 suggests other applications of our QBF-based reduction technique and closes the paper.

2 Preliminaries

In this section we recall the basic terms related to TELA and QBF.

▶ **Definition 1** (TELA). *A* transition-based Emerson-Lei automaton (TELA) *is a tuple* $\mathcal{A} = (Q, M, \Sigma, \delta, q_I, \varphi)$*, where*

- Q *is a finite set of* states*,*
- M *is a finite set of* acceptance marks*,*
- Σ *is a finite* alphabet*,*
- $\delta \subseteq Q \times \Sigma \times 2^M \times Q$ *is a* transition relation*,*
- $q_I \in Q$ *is an* initial state*, and*
- φ *is the* acceptance condition *constructed according to the following abstract syntax equation, where m ranges over M.*

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid \mathsf{Inf}\, m \mid \mathsf{Fin}\, m \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi)$$

A tuple $t = (p, a, M', q) \in \delta$ is the *transition* leading from state p to state q labelled with a and acceptance marks M'. The set M' is also referred to by $\mathit{mks}(t)$. For a set of transitions $T \subseteq \delta$, let $\mathit{mks}(T) = \bigcup_{t \in T} \mathit{mks}(t)$ denote the set of marks that appear on transitions in T.

A *run* π of $\mathcal{A}$ over an infinite word $u = u_0u_1u_2 \ldots \in \Sigma^\omega$ is an infinite sequence of adjacent transitions $\pi = (q_0, u_0, M_0, q_1)(q_1, u_1, M_1, q_2) \ldots \in \delta^\omega$ where $q_0 = q_I$. Let $\mathit{inf}(\pi)$ denote the set of transitions that appear infinitely many times in π. Run π is *accepting* iff $\mathit{inf}(\pi)$ satisfies the formula φ, where a set T of transitions satisfies $\mathsf{Inf}\, m$ iff $m \in \mathit{mks}(T)$ and it satisfies $\mathsf{Fin}\, m$ iff $m \notin \mathit{mks}(T)$. The *language* of $\mathcal{A}$ is the set $L(\mathcal{A}) = \{u \in \Sigma^\omega \mid$ there is an accepting run of $\mathcal{A}$ over $u\}$. Two automata $\mathcal{A}, \mathcal{B}$ are *equivalent* if $L(\mathcal{A}) = L(\mathcal{B})$.

An acceptance formula φ is in *disjunctive normal form (DNF)* if it is a disjunction of cubes, where each cube is a conjunction of atoms of the form $\mathsf{Fin}\, m$ or $\mathsf{Inf}\, m$. Each acceptance formula can be transformed into an equivalent formula in DNF. Formula *false* corresponds to the disjunction of zero cubes and formula *true* corresponds to the cube with zero atoms.

A *path* from a state p to a state q is a finite sequence of adjacent transitions of the form $\rho = (q_0, u_0, M_0, q_1)(q_1, u_1, M_1, q_2) \ldots (q_{n-1}, u_{n-1}, M_{n-1}, q_n) \in \delta^+$ such that $p = q_0$ and $q = q_n$. A nonempty set of states $S \subseteq Q$ is called a nontrivial *strongly connected component (SCC)* if for each $p, q \in S$ there is a path from p to q. An SCC S is *maximal* if there is no SCC S' satisfying $S \subsetneq S'$. In the rest of this paper, SCC always refers to a maximal SCC.

Given a set of states $S \subseteq Q$, let $\delta_S = \delta \cap (S \times \Sigma \times 2^M \times S)$ denote the set of all transitions between states in S. Further, for each mark $m \in M$, let $\delta_m = \{t \in \delta \mid m \in \mathit{mks}(t)\}$ denote the set of all transitions marked with m. A set of transitions $T \subseteq \delta$ is called a *cycle* if there exists a path from a state p to the same state containing each transition of T at least once and no transition outside T. Finally, we assume that each TELA $\mathcal{A}$ contains only states q that are reachable from the initial state q_I (i.e., $q = q_I$ or there is a path from q_I to q) as states that are not reachable from q_I can be eliminated without any impact on $L(\mathcal{A})$. We also assume that each TELA has at least one SCC as automata without any SCC trivially describe an empty language.

In graphical representation, we often use acceptance marks ❶, ❷, $\ldots \in M$. Further, an edge Ⓟ–ⓙ$\overset{a}{–}$ⓚ→ⓠ denotes the transition $(p, a, \{$ⓙ, ⓚ$\}, q) \in \delta$.

By choosing an appropriate acceptance condition, one can easily represent many classical kinds of automata over infinite words. For example, a Büchi automaton can be represented as a TELA with the acceptance condition $\varphi = \mathsf{Inf}$ ❶ and the single mark ❶ placed on all transitions leaving the accepting states of the Büchi automaton. Further, a Rabin automaton with k acceptance pairs can be similarly represented as a TELA with acceptance condition $\varphi = (\mathsf{Fin}$ ❶ $\wedge\ \mathsf{Inf}$ ❶'$) \vee \ldots \vee (\mathsf{Fin}$ ⓚ $\wedge\ \mathsf{Inf}$ ⓚ'$)$ and marks $M = \{$❶, ❶', $\ldots$, ⓚ, ⓚ'$\}$.

Quantified Boolean formulas (QBF) are Boolean formulas extended with universal and existential quantification over propositional variables. We assume that subformulas of the form $\forall x.\psi$ and $\exists x.\psi$ do not contain another quantification of variable x inside ψ. The semantics of $\forall x.\psi$ and $\exists x.\psi$ is given by equivalences

$$\begin{aligned} \forall x.\psi &\equiv \psi[x \to \mathit{true}] \ \wedge\ \psi[x \to \mathit{false}] \\ \exists x.\psi &\equiv \psi[x \to \mathit{true}] \ \vee\ \psi[x \to \mathit{false}] \end{aligned}$$

where $\psi[x \to \rho]$ denotes the formula ψ with all occurrences of x simultaneously replaced by ρ. The equivalences imply that each QBF can be transformed into an equivalent Boolean formula. However, the size of this Boolean formula can be exponential in the size of the original QBF. Let V be the set of all propositional variables. A mapping $\mu : V \to \{0, 1\}$ is a *model* of a QBF φ iff it is a satisfying assignment of an equivalent Boolean formula. A QBF is *satisfiable* iff it has a model.

3 Construction of quantified Boolean formulas

Recall that we aim to reduce the number of acceptance marks in a given TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_I, \varphi)$ without altering its structure and language. In other words, we look for

- a set N of acceptance marks satisfying $|N| < |M|$,
- an acceptance formula ψ over N, and
- a function $\mathit{nm} : \delta \to Q \times \Sigma \times 2^N \times Q$ assigning new marks to transitions (i.e., for each $t = (p, a, M', q) \in \delta$, we assume that $\mathit{nm}(t) = (p, a, N', q)$ for some $N' \subseteq N$) such that the automaton $\mathcal{B} = (Q, N, \Sigma, \mathit{nm}(\delta), q_I, \psi)$ is equivalent to $\mathcal{A}$.

We will actually look for ψ and nm such that each run $\pi = t_0t_1t_2\ldots$ of $\mathcal{A}$ is accepting if and only if the run $\mathit{nm}(t_0)\mathit{nm}(t_1)\mathit{nm}(t_2)\ldots$ of $\mathcal{B}$ is accepting. This requirement clearly guarantees the equivalence of $\mathcal{A}$ and $\mathcal{B}$, but it is not a necessary condition for the equivalence. Indeed, there exist automata where relaxing this requirement can lead to a bigger reduction of acceptance marks (see Figure 1). However, looking for ψ and nm that preserve the acceptance of individual runs makes the problem easier as we can, for example, ignore the labelling of transitions by the elements of Σ.

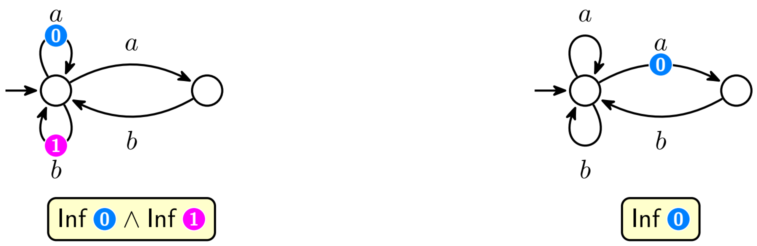


Figure 1 The left automaton accepts the words that contain infinitely many occurrences of both a and b. Each accepting run of the left automaton has to contain infinitely many occurrences of both transitions looping on the initial state. Hence, there does not exist any automaton with the same accepting runs as the left automaton and less than two acceptance marks. The right automaton accepts the same language using one acceptance mark and a different set of accepting runs.

Our reduction method is based on two facts. First, the acceptance of a run π is fully determined by $\mathit{inf}(\pi)$. Second, each set $\mathit{inf}(\pi)$ is a cycle and vice versa.

▶ **Lemma 2.** *A set $T \subseteq \delta$ is a cycle if and only if there is a run π such that $\mathit{inf}(\pi) = T$.*

Proof. To prove the direction "$\Longrightarrow$", we assume that T is a cycle. The definition says that there exists a path τ from a state p to the same state containing each transition of T at least once and no transition outside T. As our automata contain only reachable states, there exists a path ρ from the initial state q_I to p or $p = q_I$ and we set $\rho = \varepsilon$. The infinite sequence $\pi = \rho.\tau^\omega$ is a run satisfying $\mathit{inf}(\pi) = T$.

To prove the direction "$\Longleftarrow$", we consider a run π. As $\mathit{inf}(\pi)$ is the set of transitions that appear infinitely many times in π, there has to be a suffix π' of π containing only transitions of $\mathit{inf}(\pi)$. Let p be the first state of π'. As each transition of π' appears infinitely many times in π and thus also in π', there has to be a finite prefix ρ of π' such that ρ is a path from p to p that contains all transitions of $\mathit{inf}(\pi)$. In other words, the set $\mathit{inf}(\pi)$ is a cycle. ◀

Hence, our goal can be reformulated as follows. We look for a new acceptance formula ψ and a function nm such that for each cycle $T \subseteq \delta$, it holds that T satisfies φ if and only if $\mathit{nm}(T)$ satisfies ψ. This can be roughly denoted by the formula

$$\forall T \subseteq \delta \;.\; \mathit{cycle}(T) \implies \big(\mathit{satisfies}_\varphi(T) \iff \mathit{satisfies}_\psi(\mathit{nm}(T))\big).$$

In fact, this corresponds to the shape of the QBF we will construct. As we are looking for ψ and nm such that the formula holds, the subformula $\mathit{satisfies}_\psi(\mathit{nm}(T))$ contains many free variables representing possible instances of ψ and nm. If the formula is satisfiable, then each of its models encodes a desired instance of ψ and nm. In the following, we assume that we are looking for a new acceptance formula ψ in DNF. The choice of DNF is not fundamental, but inherited from our previous attempt to reduce acceptance formulas. The presented method can be easily adapted to look for ψ in *conjunctive normal form (CNF)* or in a different shape.

Now we describe the construction of the QBF in detail. The construction is parameterized by two integers $C, K \geq 0$, where K is the desired number of acceptance marks and C is the number of cubes of ψ. Without loss of generality, we assume that the reduced automaton will use the acceptance marks $N_K = \{1, 2, \ldots, K\}$. We start with a description of Boolean variables used in the constructed QBF.

- For each transition $t \in \delta$, variable e_t says whether t is in the current set T or not.

$$e_t = \begin{cases} 1 & \text{if } t \in T \\ 0 & \text{otherwise} \end{cases}$$

- For each transition $t \in \delta$ and acceptance mark $k \in N_K$, variable $n_{t,k}$ says whether k is on the transition $nm(t)$ or not.

$$n_{t,k} = \begin{cases} 1 & \text{if } k \in \mathit{mks}(nm(t)) \\ 0 & \text{otherwise} \end{cases}$$

- For each $c \in \{1, 2, \ldots, C\}$ and acceptance mark $k \in N_K$, variables $i_{c,k}$ and $f_{c,k}$ say whether the c^{th} cube of ψ contains atoms $\mathsf{Inf}\, k$ or $\mathsf{Fin}\, k$, respectively.

$$i_{c,k} = \begin{cases} 1 & \text{if the } c^{\text{th}} \text{ cube of } \psi \text{ contains } \mathsf{Inf}\, k \\ 0 & \text{otherwise} \end{cases}$$

$$f_{c,k} = \begin{cases} 1 & \text{if the } c^{\text{th}} \text{ cube of } \psi \text{ contains } \mathsf{Fin}\, k \\ 0 & \text{otherwise} \end{cases}$$

By $\vec{e}, \vec{n}, \vec{i}, \vec{f}$ we denote the vectors of all variables of the form e_t, $n_{t,k}$, $i_{c,k}$, and $f_{c,k}$, respectively. The constructed QBF have the form

$$\Phi_{C,K}(\vec{n}, \vec{i}, \vec{f}) \quad = \quad \forall \vec{e}\ .\ \mathit{cycle}(\vec{e}) \implies \big(\mathit{satisfies}_{\varphi}(\vec{e}) \iff \mathit{satisfies}_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})\big),$$

where $\forall \vec{e}$ denotes the sequence composed of $\forall e_t$ for all variables e_t. Now we define the subformulas $\mathit{satisfies}_{\varphi}(\vec{e})$, $\mathit{satisfies}_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$, and three versions of $\mathit{cycle}(\vec{e})$.

The subformula $\mathit{satisfies}_{\varphi}(\vec{e})$ says whether T satisfies the original acceptance formula φ and it is derived directly from φ. Recall that T satisfies $\mathsf{Inf}\, m$ iff $m \in \mathit{mks}(T)$, which means that T contains some transition with mark m. As the transitions with mark m form the set δ_m, $\mathsf{Inf}\, m$ can be expressed by $\bigvee_{t \in \delta_m} e_t$. Similarly, T satisfies $\mathsf{Fin}\, m$ iff $m \notin \mathit{mks}(T)$, which can be expressed by $\bigwedge_{t \in \delta_m} \neg e_t$. Hence, $\mathit{satisfies}_{\varphi}(\vec{e})$ arises from φ by replacing

- all atoms of the form $\mathsf{Inf}\, m$ by $\bigvee_{t \in \delta_m} e_t$ and
- all atoms of the form $\mathsf{Fin}\, m$ by $\bigwedge_{t \in \delta_m} \neg e_t$.

Next, we construct the subformula $\mathit{satisfies}_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$ that evaluates to true iff $nm(T)$ satisfies ψ. The subformula reflects the basic structure of ψ. As we assume that ψ is a disjunction of C cubes, we have

$$\mathit{satisfies}_{C,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f}) = \bigvee_{c \in \{1,2,\ldots,C\}} \xi_{c,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$$

where each $\xi_{c,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f})$ corresponds to one cube. Recall that the presence of atoms $\mathsf{Inf}\, k$ and $\mathsf{Fin}\, k$ in the c^{th} cube is given by variables $i_{c,k}$ and $f_{c,k}$, respectively. $\mathsf{Inf}\, k$ is satisfied by $nm(T)$ iff T contains a transition t such that $k \in \mathit{mks}(nm(t))$, which can be expressed as $\bigvee_{t \in \delta}(e_t \land n_{t,k})$. Similarly, $\mathsf{Fin}\, k$ is satisfied by $nm(T)$ iff there is no transition $t \in T$ such that $k \in \mathit{mks}(nm(t))$, which can be expressed as $\bigwedge_{t \in \delta} \neg(e_t \land n_{t,k})$. Hence, we set

$$\xi_{c,K}(\vec{e}, \vec{n}, \vec{i}, \vec{f}) = \bigwedge_{k \in N_K} \Big(i_{c,k} \implies \bigvee_{t \in \delta}(e_t \land n_{t,k})\Big) \land \Big(f_{c,k} \implies \bigwedge_{t \in \delta} \neg(e_t \land n_{t,k})\Big).$$

It remains to define the subformula $\mathit{cycle}(\vec{e})$. Let $T_{\vec{e}}$ denote the set of transitions represented by $\vec{e}$. The original intended meaning of $\mathit{cycle}(\vec{e})$ is

$$\mathit{cycle}(\vec{e}) \iff T_{\vec{e}} \text{ is a cycle.}$$

In fact, only the direction "$\Longleftarrow$" is needed for the correctness of our reduction method. If there are some valuations of $\vec{e}$ such that $\mathit{cycle}(\vec{e})$ holds and $T_{\vec{e}}$ is not a cycle, then we will superfluously require the equivalence $\mathit{satisfies}_\varphi(\vec{e}) \iff \mathit{satisfies}_{C,K}(\vec{e},\vec{n},\vec{i},\vec{f})$ on these valuations. These superfluous constraints can lead to loss of reduction opportunities, but not to incorrectness. This observation allows us to trade the precision of $\mathit{cycle}(\vec{e})$ for its simplicity.

We define three versions of $\mathit{cycle}(\vec{e})$:

- $\mathit{cycle}_1(\vec{e})$ is a lightweight version, which only says that $T_{\vec{e}}$ is nonempty and $T_{\vec{e}} \subseteq \delta_S$ for some SCC S. Except for SCCs, it does not use the information about the automaton structure, but it comes with an interesting simplification of the whole formula $\Phi_{C,K}$.
- $\mathit{cycle}_2(\vec{e})$ is an intermediate version. It says that $T_{\vec{e}}$ is nonempty, $T_{\vec{e}} \subseteq \delta_S$ for some SCC S, and every transition in $T_{\vec{e}}$ has a preceding and a succeeding transition in $T_{\vec{e}}$, which is a necessary condition for being a cycle, but not a sufficient one.
- $\mathit{cycle}_3(\vec{e})$ is a strict version saying that $T_{\vec{e}}$ is a cycle. Unfortunately, it uses additional universally quantified variables corresponding to automata states. Transformation of $\Phi_{C,K}$ to prenex normal form turns the quantifiers to existential ones and the resulting formula thus contains quantifier alternation.

We write $\Phi_{j,C,K}$ when we want to emphasize that a particular formula $\Phi_{C,K}$ contains the version $\mathit{cycle}_j(\vec{e})$.

3.1 Lightweight version $\mathit{cycle}_1(\vec{e})$

The lightweight version is defined as

$$\mathit{cycle}_1(\vec{e}) = \bigvee_{\text{SCC } S} \Big(\bigvee_{t\in\delta_S} e_t \;\wedge \bigwedge_{t'\in\delta\smallsetminus\delta_S} \neg e_{t'} \Big)$$

which means only that $T_{\vec{e}}$ is nonempty and $T_{\vec{e}} \subseteq \delta_S$ for some SCC S. This condition is satisfied by every cycle.

The formula $\Phi_{1,C,K}$ built with $\mathit{cycle}_1(\vec{e})$ says that for every nonempty set $T \subseteq \delta_S$ where S is an SCC, T satisfies φ if and only if $\mathit{nm}(T)$ satisfies ψ. Note that the only aspects of a transition t reflected by the formula are its set of marks $\mathit{mks}(t)$ and its affiliation to an SCC. Hence, we do not have to distinguish between transitions that are affiliated to the same SCC and have the same sets of marks.

Let us now fix an SCC S. We define an equivalence $\sim_S \subseteq \delta_S \times \delta_S$ on transitions such that $t_1 \sim_S t_2$ whenever $\mathit{mks}(t_1) = \mathit{mks}(t_2)$.

▶ **Lemma 3.** *Assume that there is a function nm and a formula ψ such that*

$$\textit{for every set } \emptyset \neq T \subseteq \delta_S \textit{ it holds } (T \textit{ satisfies } \varphi \iff \mathit{nm}(T) \textit{ satisfies } \psi). \tag{1}$$

Then there exists a function nm' that respects the equivalence $\sim_S$ (i.e., it assigns the same marks to equivalent transitions) and

$$\textit{for every set } \emptyset \neq T \subseteq \delta_S \textit{ it holds } (T \textit{ satisfies } \varphi \iff \mathit{nm}'(T) \textit{ satisfies } \psi). \tag{2}$$

Proof. Let nm be a function and ψ a formula such that (1) holds. To construct the function nm', we first select one transition from each equivalence class of $\sim_S$. For every transition $t = (p, a, M', q) \in \delta_S$, by $\bar{t}$ we denote the selected transition equivalent to t and we define the function nm' such that $\mathit{nm}'(t) = (p, a, \mathit{mks}(\mathit{nm}(\bar{t})), q)$. Note that we do not need to discuss the value of nm' on transitions outside δ_S as it is not relevant for the lemma. Clearly, nm' respects the equivalence $\sim_S$. It remains to show that (2) holds for nm' and ψ.

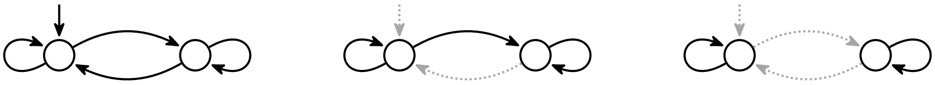


Figure 2 An automaton structure (left) and two sets $T_{\vec{e}}$ (middle and right) that are not cycles even if $cycle_2(\vec{e})$ holds. The transition labels and acceptance marks are not depicted.

Let $T \subseteq \delta_S$ be a nonempty set. We construct the set $\overline{T} = \{\overline{t} \mid t \in T\}$. As $mks(t) = mks(\overline{t})$ for all transitions of δ_S, we get $mks(T) = mks(\overline{T})$ and thus

$$T \textit{ satisfies } \varphi \iff \overline{T} \textit{ satisfies } \varphi.$$

Now we apply (1) to $\overline{T}$ and we get

$$\overline{T} \textit{ satisfies } \varphi \iff nm(\overline{T}) \textit{ satisfies } \psi.$$

Finally, the definition of nm' implies that $nm'(T) = nm(\overline{T})$ and thus

$$nm(\overline{T}) \textit{ satisfies } \psi \iff nm'(T) \textit{ satisfies } \psi.$$

Altogether, we obtain

$$T \textit{ satisfies } \varphi \iff \overline{T} \textit{ satisfies } \varphi \iff nm(\overline{T}) \textit{ satisfies } \psi \iff nm'(T) \textit{ satisfies } \psi$$

which proves that (2) holds for nm' and ψ. ◀

The lemma suggests the following simplification of the whole formula $\Phi_{1,C,K}$ built with $cycle_1(\vec{e})$. Before we build the formula, we compute the equivalences $\sim_S$ for all SCCs and temporarily remove all transitions affiliated to SCCs except one of each equivalence class. Then we build the formula $\Phi_{1,C,K}$ for the pruned automaton. The more transitions we removed, the shorter formula with less e_t variables we obtain. If the formula $\Phi_{1,C,K}$ for the pruned automaton is satisfiable, we derive nm and ψ from its model and extend nm to all transitions of the original automaton such that it changes the acceptance marks on all equivalent transitions in the same way. In the following, we use this simplification whenever $\Phi_{1,C,K}$ is employed.

3.2 Intermediate version $cycle_2(\vec{e})$

The intermediate version says that $T_{\vec{e}}$ is nonempty, $T_{\vec{e}} \subseteq \delta_S$ for some SCC S, and for each state $q \in Q$ it holds that $T_{\vec{e}}$ contains a transition leading to q if and only if it contains a transition leaving q. Formally,

$$cycle_2(\vec{e}) = cycle_1(\vec{e}) \;\wedge\; \bigwedge_{q \in Q} \Big(\bigvee_{t' \in \delta \,\cap\, Q \times \Sigma \times 2^M \times \{q\}} e_{t'} \iff \bigvee_{t'' \in \delta \,\cap\, \{q\} \times \Sigma \times 2^M \times Q} e_{t''} \Big).$$

This condition is satisfied by every cycle, but also by some sets of transitions that are not cycles. Some examples of such sets are provided in Figure 2.

3.3 Strict version $cycle_3(\vec{e})$

Before we give the definition of $cycle_3(\vec{e})$, we prove that cycles can be characterised in the following way.

▶ **Lemma 4.** *A nonempty set $T \subseteq \delta$ is a cycle if and only if, for each set of states $S \subseteq Q$, one of the following conditions holds.*

A. *All transitions in T lead from a state in S to a state in S (i.e., $T \subseteq \delta_S$).*

B. *All transitions in T lead from a state outside S to a state outside S (i.e., $T \subseteq \delta_{Q \smallsetminus S}$).*

C. *T contains a transition leading from a state in S to a state outside S and a transition leading from a state outside S to a state in S.*

Proof. We first prove the direction "$\Longrightarrow$". Let T be a cycle and $S \subseteq Q$ be an arbitrary set of states. We show that if (A) and (B) do not hold, then (C) has to hold. Hence, assume that $T \nsubseteq \delta_S$ and $T \nsubseteq \delta_{Q \smallsetminus S}$. Then there are two cases.

- T contains a transition $t \in \delta_S$ and a transition $t' \in \delta_{Q \smallsetminus S}$. The definition of a cycle implies that there exists a path $t_1 t_2 \dots t_n \in T^+$ from a state p back to p containing both t and t'. However, this implies that T contains a transition leading from a state in S to a state outside S and a transition leading from a state outside S to a state in S.
- T contains a transition t leading from a state in S to a state outside S (or vice versa). However, as T is a cycle, there exists a path $t_1 t_2 \dots t_n \in T^+$ that leads from a state p to the same state and contains t. Hence, T has to contain also a transition leading from a state outside S to a state in S (or vice versa).

In both cases, (C) holds.

Now we prove the opposite direction "$\Longleftarrow$" by contraposition. Assume that a nonempty set T is not a cycle. We show that there is a set $S \subseteq Q$ such that neither (A) nor (B) nor (C) holds. Let p be a state such that some transition of T leads from p. We define the set S_{post} of states reachable from p via transitions in T and the set S_{pre} of states from which p is reachable via transitions of T.

$$\begin{aligned} S_{post} &= \{p\} \cup \{q \in Q \mid \text{there is a path in } T^+ \text{ from } p \text{ to } q\} \\ S_{pre} &= \{p\} \cup \{q \in Q \mid \text{there is a path in } T^+ \text{ from } q \text{ to } p\} \end{aligned}$$

As there is a transition of T leading from p, we have that $T \nsubseteq \delta_{Q \smallsetminus S_{post}}$ and $T \nsubseteq \delta_{Q \smallsetminus S_{pre}}$, i.e., (B) does not hold for S_{post} and S_{pre}. Further, the definition of S_{post} implies that there is no transition of T leading from a state in S_{post} to a state outside S_{post}, which means that (C) does not hold for S_{post}. Similarly, T contains no transition leading from a state outside S_{pre} to a state in S_{pre}, which means that (C) does not hold for S_{pre}. Now we prove by contradiction that (A) does not hold for at least one of S_{post}, S_{pre}. Hence, let us assume that $T \subseteq \delta_{S_{post}}$ and $T \subseteq \delta_{S_{pre}}$. Then for each $t_i \in T$ leading from p_i to q_i we have that $p_i \in S_{post}$ and $q_i \in S_{pre}$, which implies that

- $p_i = p$ (we set $\rho'_i = \varepsilon$ in this case) or there is a path $\rho'_i \in T^+$ leading from p to p_i, and
- $q_i = p$ (we set $\rho''_i = \varepsilon$ in this case) or there is a path $\rho''_i \in T^+$ leading from q_i to p.

Then there is a path $\rho_i = \rho'_i t_i \rho''_i \in T^+$ leading from p back to p and containing t_i. If we concatenate all these paths, we get the path $\rho_1 \rho_2 \dots \rho_{|T|} \in T^+$ that contains all transitions of T and leads from p back to p, which means that T is a cycle. This is a contradiction. ◀

The formula $\mathit{cycle}_3(\vec{e})$ says that $T_{\vec{e}}$ is nonempty and each set $S \subseteq Q$ satisfies (A) or (B) or (C). For each state $q \in Q$, variable s_q says whether q is in the current set S or not.

$$s_q = \begin{cases} 1 & \text{if } q \in S \\ 0 & \text{otherwise} \end{cases}$$

By $\vec{s}$ we denote the vectors of all variables of the form s_q. The formula $\mathit{cycle}_3(\vec{e})$ is defined as follows.

$$\begin{aligned}
\mathit{cycle}_3(\vec{e}) &= \bigvee_{t\in\delta} e_t \ \wedge\ \forall\vec{s}\ .\ \zeta_A(\vec{e},\vec{s}) \vee \zeta_B(\vec{e},\vec{s}) \vee \zeta_C(\vec{e},\vec{s})\\
\zeta_A(\vec{e},\vec{s}) &= \bigwedge_{t=(p,a,M',q)\in\delta} \big(e_t \implies (s_p \wedge s_q)\big)\\
\zeta_B(\vec{e},\vec{s}) &= \bigwedge_{t=(p,a,M',q)\in\delta} \big(e_t \implies (\neg s_p \wedge \neg s_q)\big)\\
\zeta_C(\vec{e},\vec{s}) &= \Big(\bigvee_{t=(p,a,M',q)\in\delta} e_t \wedge s_p \wedge \neg s_q\Big) \wedge \Big(\bigvee_{t=(p,a,M',q)\in\delta} e_t \wedge \neg s_p \wedge s_q\Big)
\end{aligned}$$

3.4 Complexity of formulas

The constructed formulas $\Phi_{j,C,K}$ for $j \in \{1,2,3\}$ use $|\delta|$ universally quantified variables e_t, $|\delta|\cdot K$ free variables $n_{t,k}$, and $C\cdot K$ free variables $i_{c,k}$ and $f_{c,k}$. The formula $\Phi_{3,C,K}$ additionally uses $|Q|$ variables s_q that are existentially quantified (when the formula is transformed to prenex normal form) in the scope of universal quantification of variables e_t.

To analyze the length of the formulas, we start with its subformulas. One can easily check that $|\mathit{satisfies}_\varphi(\vec{e})| \in \mathcal{O}(|\varphi|\cdot|\delta|)$ and $|\mathit{satisfies}_{C,K}(\vec{e},\vec{n},\vec{i},\vec{f})| \in \mathcal{O}(C\cdot K\cdot|\delta|)$. Further, $|\mathit{cycle}_1(\vec{e})|, |\mathit{cycle}_2(\vec{e})| \in \mathcal{O}(S\cdot|\delta|)$, where S is the number of SCCs in the automaton. Next, $|\mathit{cycle}_3(\vec{e})| \in \mathcal{O}(|\delta|+|Q|)$, which can be simplified to $|\mathit{cycle}_3(\vec{e})| \in \mathcal{O}(|\delta|)$ as $|Q|\leq|\delta|$ follows from the assumptions that all states are reachable and each automaton has at least one SCC. Altogether, we get $|\Phi_{1,C,K}|, |\Phi_{2,C,K}| \in \mathcal{O}(S\cdot|\delta|+|\varphi|\cdot|\delta|+C\cdot K\cdot|\delta|) = \mathcal{O}\big((S+|\varphi|+C\cdot K)\cdot|\delta|\big)$ and $|\Phi_{3,C,K}| \in \mathcal{O}(|\delta|+|\varphi|\cdot|\delta|+C\cdot K\cdot|\delta|) = \mathcal{O}\big((|\varphi|+C\cdot K)\cdot|\delta|\big)$. Note that the formula $\Phi_{3,C,K}$ is asymptotically shorter than $\Phi_{1,C,K}$ and $\Phi_{2,C,K}$, but it contains an additional quantifier alternation.

3.5 Optimizations of formulas

Finally, we mention three simple optimizations of the formula construction, which are always applied in the rest of the paper.

The first optimization is based on the fact that every cycle is completely included in the transition set δ_S of some SCC S. Hence, all transitions t that do not lead between states of the same SCC can be completely ignored during the formula construction. The acceptance marks on such a transition t do not affect the acceptance of any run as t appears at most once on each run. For these transitions t, we can define $\mathit{nm}(t)$ such that $\mathit{mks}(\mathit{nm}(t)) = \emptyset$.

The second optimization is specific for $\Phi_{3,C,K}$. In the construction of $\mathit{cycle}_3(\vec{e})$, we replace the subformula $\bigvee_{t\in\delta} e_t$ enforcing the nonemptiness of $T_{\vec{e}}$ by $\mathit{cycle}_2(\vec{e})$. This modification prolongs the formula, but it does not change the overall semantics of $\mathit{cycle}_3(\vec{e})$ and our preliminary experiments showed that QBF solvers can often solve the modified formula $\Phi_{3,C,K}$ faster.

The third optimization extends $\Phi_{j,C,K}$ into the conjunction

$$\Phi_{j,C,K} \ \wedge \bigwedge_{c\in\{1,2,\dots,C\}} \bigwedge_{k\in N_K} (\neg i_{c,k} \vee \neg f_{c,k}).$$

The added part says that no cube contains both $\mathsf{Inf}\,k$ and $\mathsf{Fin}\,k$ for any k. A cube with both $\mathsf{Inf}\,k$ and $\mathsf{Fin}\,k$ would be useless as it cannot be satisfied by any run.

Algorithm 1 The single-level reduction procedure.

```
Procedure SingleLevelReduction($\mathcal{A}, j, reduceC$)
    Input: TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_I, \varphi)$, $j \in \{1,2,3\}$, $reduceC \in \{true, false\}$
    Output: an equivalent TELA with the same structure as $\mathcal{A}$ and with at most as
            many acceptance marks as in $\mathcal{A}$

    $C_\mathcal{A} \leftarrow$ the number of cubes in the formula $\varphi$ transformed to DNF
    $K_\mathcal{A} \leftarrow$ the number of acceptance marks in $\mathcal{A}$
    $C \leftarrow C_\mathcal{A}$
    $K \leftarrow K_\mathcal{A}$
    while $K > 1 \ \wedge \ satisfiable(\Phi_{j,C,K-1})$ do $K \leftarrow K-1$
    if $K = 1$ then
        if all cycles in $\mathcal{A}$ are accepting then          // check the condition true
            return $(Q, \emptyset, \Sigma, \delta', q_I, true)$ where $\delta'$ is $\delta$ with all marks removed
        if all cycles in $\mathcal{A}$ are rejecting then          // check the condition false
            return $(Q, \emptyset, \Sigma, \delta', q_I, false)$ where $\delta'$ is $\delta$ with all marks removed
    if reduceC then                              // reduction of the number of cubes
        while $C > 1 \ \wedge \ satisfiable(\Phi_{j,C-1,K})$ do $C \leftarrow C-1$
    if $K < K_\mathcal{A} \ \vee \ C < C_\mathcal{A}$ then
        compute $nm$ and $\psi$ from a model of $\Phi_{j,C,K}$
        return $(Q, N_K, \Sigma, nm(\delta), q_I, \psi)$
    return $\mathcal{A}$
```

We have also made some experiments with breaking the symmetries in the formula models. In particular, we have ordered new acceptance marks by their placements on transitions and we have ordered the cubes by their content. As the effect of these modifications was inconclusive, we do not describe it here.

4 Reduction algorithm

This section explains how we use the QBF constructed in the previous section to reduce the number of acceptance marks in TELA. First, we describe a *single-level* reduction, which uses only a single kind of QBF. More precisely, we talk about *level 1*, *level 2*, or *level 3* reduction when $\Phi_{1,C,K}$, $\Phi_{2,C,K}$, or $\Phi_{3,C,K}$ is used, respectively.

The reduction procedure called *SingleLevelReduction* is given in Algorithm 1. Besides the reduction of acceptance marks, the algorithm also reduces the number of cubes in the acceptance formula if the last argument *reduceC* is set to *true*. The first **while** loop gradually decreases the number of marks until $K = 1$ is reached or the QBF solver behind the function *satisfiable*($\Phi_{j,C,K-1}$) fails to reduce the number of marks, i.e., it claims unsatisfiability of the formula or it runs out of resources. If the loop ends with $K = 1$, we check whether an acceptance condition without any mark (i.e., *true* or *false*) can be used. These checks are based on an inspection of the automaton rather than on QBF solving. If some of the checks succeeds, we return the corresponding automaton without any acceptance mark. Otherwise, if *reduceC* is set to *true* then the procedure gradually reduces the number of cubes in the second **while** loop. Note that the loop never checks for acceptance condition with 0 cubes as it is equivalent to *false* and this case was treated above. Finally, if the procedure succeeds to reduce the number of marks or cubes, it constructs the modified automaton. Otherwise, it returns the original automaton.

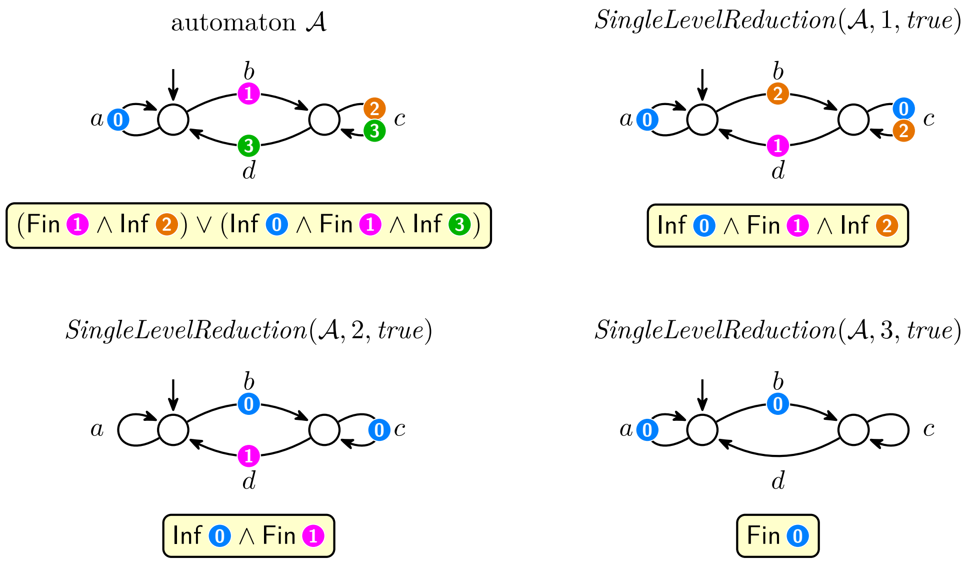

automaton $\mathcal{A}$
a 0
b 1
c 2 3
d 3
(Fin 1 ∧ Inf 2) ∨ (Inf 0 ∧ Fin 1 ∧ Inf 3)
SingleLevelReduction($\mathcal{A}$, 1, *true*)
a 0
b 2
c 0 2
d 1
Inf 0 ∧ Fin 1 ∧ Inf 2
SingleLevelReduction($\mathcal{A}$, 2, *true*)
a
b 0
c 0
d 1
Inf 0 ∧ Fin 1
SingleLevelReduction($\mathcal{A}$, 3, *true*)
a 0
b 0
c
d
Fin 0

Figure 3 An example illustrating the results of the three single-level reductions: an input automaton $\mathcal{A}$ and the automata obtained by reducing it with level 1, level 2, and level 3.

The algorithm can be reformulated to use an incremental approach instead of building a new formula in each iteration of the **while** loops. The incremental version of the first **while** loop builds the formula $\Phi = \Phi_{j,C,K-1}$ only in the first iteration. In each subsequent iteration, it extends this formula with a condition saying that one more mark is not used in the automaton, i.e., the mark is neither on edges, nor in the acceptance formula. For example, if we want to say that the mark $k \in N_K$ is not used, we replace Φ by

$$\Phi \;\wedge\; \bigwedge_{t\in\delta} \neg n_{t,k} \;\wedge \bigwedge_{c\in\{1,2,\ldots C\}} (\neg i_{c,k} \wedge \neg f_{c,k}).$$

The second **while** loop can be transformed to an incremental version similarly. The incremental approach benefits from the fact that some QBF solvers can decide an extended formula faster as they reuse the information computed when solving the original formula.

Figure 3 shows a very simple automaton $\mathcal{A}$ and the three automata produced by calls of *SingleLevelReduction*($\mathcal{A}$, j, *true*) for $j \in \{1, 2, 3\}$. The figure clearly illustrates that the higher level of reduction we use, the more acceptance marks can be reduced. On the other side, lower levels are typically faster. The best results can be often achieved by combining reductions of all levels. We call this approach *multi-level* reduction. It is a straightforward sequential application of the three levels, see Algorithm 2.

5 Implementation

The presented reduction algorithms have been implemented in a tool called `telatko`. It is implemented in Python 3 and uses the Spot library [9] for automata parsing and manipulation, and the theorem prover Z3 [8] to solve the satisfiability of QBF transformed to prenex (non-CNF) normal form. Our tool is available at

`https://gitlab.fi.muni.cz/xschwar3/telatko`

under the GNU GPLv3 license. The tool can be executed by the command

```
telatko -F <input.hoa> [-L j] [-C] [-I] [-T t] [-O <output.hoa>]
```

Algorithm 2 The multi-level reduction procedure.

```
Procedure MultiLevelReduction($\mathcal{A}$, reduceC)
    Input: TELA $\mathcal{A} = (Q, M, \Sigma, \delta, q_I, \varphi)$ and $reduceC \in \{true, false\}$
    Output: an equivalent TELA with the same structure as $\mathcal{A}$ and with at most as
            many acceptance marks as in $\mathcal{A}$

    $\mathcal{A} \leftarrow SingleLevelReduction(\mathcal{A}, 1, false)$
    $\mathcal{A} \leftarrow SingleLevelReduction(\mathcal{A}, 2, false)$
    $\mathcal{A} \leftarrow SingleLevelReduction(\mathcal{A}, 3, reduceC)$
    return $\mathcal{A}$
```

where

`-F <input.hoa>` specifies the file with the input automaton in HOA format [3],

`-L` j specifies the reduction level; if omitted, the multi-level reduction is used,

`-C` switches on the reduction of the number of cubes after the number of marks is reduced (it corresponds to $reduceC = true$ in Algorithms 1 and 2),

`-I` switches on the incremental version,

`-T` t sets the timeout for each QBF query to t seconds (the default value is 50 seconds),

`-O <output.hoa>` specifies the output file; if omitted, the produced automaton is sent to *stdout* in the HOA format.

If some call of the function $satisfiable(\Phi_{j,C,K-1})$ in the first **while** loop of Algorithm 1 does not return *true*, then the name of the output automaton (included in the generated HOA) encodes the reason for it. In the case of a single level reduction, the name has the form `L`j`_`k`_`X, where j is the considered level, $k = K - 1$ is the number of acceptance marks considered by the formula, X is either `U` if the formula is unsatisfiable or `T` if the solver did not decide within the time limit. If X is `T`, a longer timeout may lead to further reductions. If the multi-level reduction is used, the automaton name contains the information from all levels. For example, the name '`L1_5_U L2_3_U L3_1_T`' means that level 1 reduced the number of marks to 6 (reduction to 5 is impossible on this level), level 2 reduced it to 4, and level 3 to 2 as the QBF solver did not finish in the time limit when trying to reduce the number of marks to 1.

6 Experimental evaluation

To evaluate our reduction technique, we applied `telatko` to automata produced by the following process. We started with two sets of LTL formulas.

- One set contains all LTL formulas from *literature* that are provided by the tool `genltl` of the Spot library [9] 2.10.4. For parameterized formula patterns, we consider instances for all combinations of parameter values from 1 to 4.
- The second set consists of 400 random LTL formulas with 4 atomic propositions. These formulas were generated by the tool `randltl` of the Spot library.

On both these sets, we applied the tool `ltlfilt` of the Spot library to simplify the formulas and remove duplicates and formulas equivalent to *true* and *false*. After these steps, we had 348 LTL formulas from literature and 335 random formulas. Formulas from both sets have been translated to nondeterministic TELA by two state-of-the-art translators, namely `ltl2tgba` (used with option `-G` to get generic TELA) from the Spot library [9] and `ltl3tela` [15], and to deterministic TELA by `ltl3tela` with option `-D1` and by two

Table 1 Considered translators and the numbers of *fails* and successfully constructed automata with *at most 1 mark* and with *at least 2 marks* for each translator and set of formulas.

		348 formulas from literature			335 random formulas		
			automata with			automata with	
translator	(version)$^{\text{web}}$	fails	at most 1 mark	**at least 2 marks**	fails	at most 1 mark	**at least 2 marks**
`ltl2tgba -G`	(2.10.4)[1]	0	278	**70**	0	320	**15**
`ltl3tela`	(2.2.0)[2]	18	239	**91**	0	286	**49**
`ltl3tela -D1`	(2.2.0)[2]	20	247	**81**	0	291	**44**
`ltl2dela`	(21.0)[3]	5	214	**129**	0	246	**89**
`ltl2dgra`	(21.0)[3]	12	102	**234**	0	130	**205**

state-of-the-art translators from the Owl library [14], namely `ltl2dela` (known as Delag) [16] and `ltl2dgra` (known as Rabinizer 4) [13]. Some translators failed on some formulas: they usually reached a timeout of 60 seconds or produced an automaton that cannot be parsed by the Spot library. Further, we have removed automata with 0 or 1 acceptance mark as there is a little point in reducing these. Table 1 shows the exact versions of the translators. For each translator and each set of formulas, the table also provides the number of fails, the number of produced automata with less than two marks, and the number of automata with at least two marks. The numbers of automata with at least two marks are typeset in bold as these automata are actually used for the experimental evaluation of our reduction technique.

To all automata, we have applied all single-level reductions and the multi-level reduction, always with incremental approach and without reducing the number of cubes. We do not reduce the number of cubes as our primary aim is to reduce the number of acceptance marks. The timeout for each QBF query was set to 30 seconds. All reductions have been performed by the tool `telatko` built with Spot library version 2.10.4 and Z3 version 4.8.15. The experiments have been run on a computer with Intel® Core™ i7-8700 processor and 32 GB of memory running Ubuntu 20.04.4. We used the tool `autcross` of the Spot library to get the statistics of the reduced automata and the running times.

For each automata set identified by the translator and the set of formulas, Table 2 shows the cumulative numbers of marks in the input automata set and after each reduction, together with the reduction ratio and total time spent by the considered reduction. The column *solver timeout* shows the number of automata for which the last query to QBF solver did not finish within the 30 seconds limit. The timeout of the last QBF query means that the automaton may be potentially further reduced if a longer time limit is used. One can observe that a higher level sometimes achieves a smaller reduction than a lower level (e.g., compare level 1 and level 2 for `ltl3tela` on automata coming from formulas from literature). This is caused by the QBF solver timeouts occurring earlier as formulas constructed by the higher level are more complex. The automata sets produced by `ltl2dela` and `ltl2dgra` on formulas from literature do not contain any automaton where level 2 or level 3 achieves a better result than level 1. However, all levels contribute to the reductions in the multi-level setting.

[1] `https://spot.lrde.epita.fr`
[2] `https://github.com/jurajmajor/ltl3tela`
[3] `https://owl.model.in.tum.de/`

Table 2 For each automata set identified by the translator and the set of formulas, the table provides the cumulative number of acceptance marks before any reduction (in the box), after reduction of individual levels and after multi-level reduction (column *marks*). The column *reduction* shows the percentage of saved acceptance marks and *time* reports the cumulative reduction time in seconds. The column *solver timeout* indicates the number of instances where the last call to the QBF solver timed out.

translator	reduction level	reduction of marks in automata from formulas from literature				reduction of marks in automata from random formulas			
		marks	reduction [%]	time [s]	solver timeout	marks	reduction [%]	time [s]	solver timeout
`ltl2tgba -G`		198 marks in 70 automata				32 marks in 15 automata			
	1	198	0.0	48.5	0	32	0.0	8.4	0
	2	198	0.0	65.2	0	31	3.1	9.4	0
	3	189	4.5	409.8	7	26	18.8	43.9	1
	multi	189	4.5	427.6	7	26	18.8	44.9	1
`ltl3tela`		348 marks in 91 automata				120 marks in 49 automata			
	1	332	4.6	530.3	13	101	15.8	32.4	0
	2	334	4.0	551.0	14	100	16.7	32.3	0
	3	326	6.3	698.5	18	95	20.8	66.9	1
	multi	319	8.3	1619.2	18	95	20.8	73.4	1
`ltl3tela -D1`		272 marks in 81 automata				97 marks in 44 automata			
	1	272	0.0	383.1	9	95	2.1	23.8	0
	2	272	0.0	386.6	10	95	2.1	24.6	0
	3	272	0.0	950.2	14	92	5.2	54.1	0
	multi	272	0.0	1659.6	16	92	5.2	67.2	1
`ltl2dela`		523 marks in 129 automata				234 marks in 89 automata			
	1	386	26.2	811.4	18	154	34.2	89.0	0
	2	391	25.2	1071.6	19	153	34.6	123.6	0
	3	397	24.1	7326.8	26	149	36.3	172.7	2
	multi	378	27.7	9186.0	24	148	36.8	219.8	2
`ltl2dgra`		882 marks in 234 automata				491 marks in 205 automata			
	1	544	38.3	859.1	14	293	40.3	275.6	0
	2	554	37.2	1073.5	17	280	43.0	283.9	0
	3	553	37.3	1349.7	22	267	45.6	433.6	3
	multi	535	39.3	2434.0	23	264	46.2	411.7	2

Table 3 The effect of multi-level reduction on all considered automata constructed from formulas from literature. A cell on coordinates (x, y) contains the number of automata that have been reduced from x to y acceptance marks. If the cell contains a sum of two numbers, the latter represents the number of automata where the attempt to reduce another mark has been unsuccessful due to a QBF solver timeout.

acceptance marks after the reduction	2	3	4	5	6	7	8	9	10–14	15–19	20–24
20–24											0+1
15–19										0+4	0+1
10–14									0+8	0+1	0
9								0	0	0	0
8							0+16	0+1	0	0	0
7						0	0+2	0	0+1	0	0
6					2+12	0+3	0	0+1	1	0	0
5				10+5	2	0	0+1	1	0	0	0
4			46+10	14	5+2	10	2+1	4	4	0+1	0
3		73+2	5+2	2	0	1	0	0	0+1	0	0
2	96+10	27	8+1	4	2	0	0+1	0	0	0	0
1	149	11	4	2	1	1	0	0	0	0	0
0	27	2	1	0	0	0	0	0	0	0	0

acceptance marks before the reduction

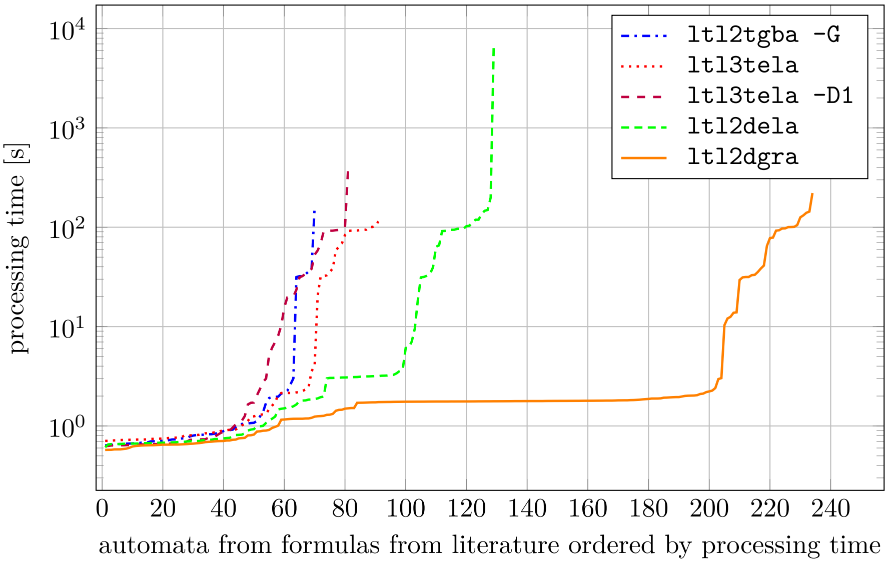
Table 4 The effect of multi-level reduction on all considered automata constructed from random formulas. The meaning of each cell is the same as in Table 3.

acceptance marks after the reduction	2	3	4	5	6	7	8	9
5				0	0+1	0	0	0
4			0	2	0+1	0	0	1
3		11+1	8	2	0	0	1	0
2	107+3	29+1	22	1	1	0	0	0
1	188	10	9	0	0	0	0	0
0	3	0	0	0	0	0	0	0

acceptance marks before the reduction

Table 3 shows the effect of multi-level reduction to the number of acceptance marks in individual automata constructed from formulas from literature. The table indicates that in many cases only 1 or 2 marks can be saved. However, the achieved reduction is substantial for some automata with a higher number of original acceptance marks. For example, in 26 cases, we have reduced 7 or more acceptance marks to only 4 or less. Table 4 shows the same information for automata constructed from random formulas.

Figure 4 presents the time spent by multi-level reduction on individual automata of each automata set. The charts show a pleasing finding that for every set, most automata are reduced in under 5 seconds and the high cumulative running times are caused by a relatively small number of complicated automata.



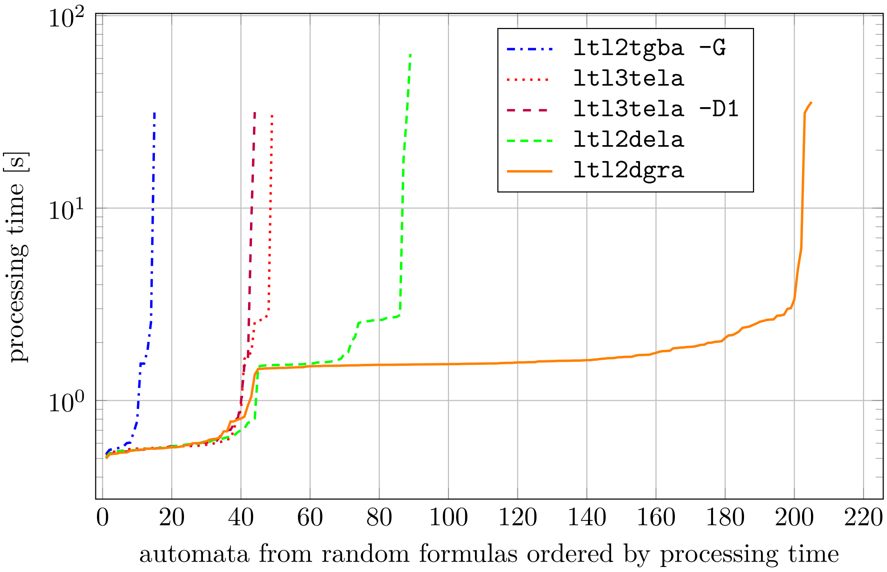


Figure 4 Running times of `telatko` on individual automata of each automata set. Automata sets constructed from formulas from literature are in the upper graph, automata sets constructed from random formulas are in the lower graph. Each line shows the time (y axis) needed by `telatko` to process the x^{th} automaton of the set, where automata in the set are ordered by their processing time.

7 Conclusions

We have presented a method reducing the number of acceptance marks in transition-based Emerson-Lei automata with use of QBF solving and without altering automata structure. We have implemented the method in a tool called `telatko`. The current applications of the tool are twofold. First, it can reduce the number of acceptance marks of a given TELA. Second, it discloses how tools producing TELA are economical with acceptance marks. The presented experimental results show that the tool can indeed reduce the number of acceptance marks in automata produced by all considered state-of-the-art LTL to automata translators. Further, it clearly shows that the translators of the Owl library are significantly less economical with acceptance marks than the other two translators.

The reduction of acceptance marks is not the only application of the presented approach. For example, it can be easily adapted to look for an equivalent automaton with the same structure and an acceptance formula of a specific form (e.g., without any $\mathsf{Fin}\,m$ atoms). Even though the QBF queries can be time-consuming, in practice one can often find a good trade-off between speed and efficiency by adjusting the formula precision and choosing a reasonable timeout.

References

1 Souheib Baarir and Alexandre Duret-Lutz. Mechanizing the minimization of deterministic generalized Büchi automata. In *Proceedings of the 34th IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems (FORTE'14)*, volume 8461 of *Lecture Notes in Computer Science*, pages 266–283. Springer, June 2014. `doi:10.1007/978-3-662-43613-4_17`.

2 Souheib Baarir and Alexandre Duret-Lutz. SAT-based minimization of deterministic ω-automata. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning – 20th International Conference, LPAR 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 79–87. Springer, 2015. `doi:10.1007/978-3-662-48899-7_6`.

3 Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi Omega-Automata Format. In *Proceedings of the 27th Conference on Computer Aided Verification (CAV'15)*, volume 8172 of *Lecture Notes in Computer Science*, pages 442–445. Springer, 2015. See also `http://adl.github.io/hoaf/`.

4 Tomáš Babiak, Thomas Badie, Alexandre Duret-Lutz, Mojmír Křetínský, and Jan Strejček. Compositional approach to suspension and other improvements to LTL translation. In Ezio Bartocci and C. R. Ramakrishnan, editors, *Model Checking Software – 20th International Symposium, SPIN 2013, Stony Brook, NY, USA, July 8-9, 2013. Proceedings*, volume 7976 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2013. `doi:10.1007/978-3-642-39176-7_6`.

5 Christel Baier, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, David Müller, and Jan Strejček. Generic emptiness check for fun and profit. In *Proceedings of the 17th International Symposium on Automated Technology for Verification and Analysis (ATVA'19)*, volume 11781 of *Lecture Notes in Computer Science*, pages 445–461. Springer, 2019. `doi:10.1007/978-3-030-31784-3_26`.

6 Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. Optimal transformations of games and automata using muller conditions. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ICALP.2021.123`.

7 Antonio Casares, Alexandre Duret-Lutz, Klara J. Meyer, Florian Renkin, and Salomon Sickert. Practical applications of the Alternating Cycle Decomposition. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part II*, volume 13244 of *Lecture Notes in Computer Science*, pages 99–117. Springer, 2022. doi:10.1007/978-3-030-99527-0_6.

8 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.

9 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 – A framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, 2016. doi:10.1007/978-3-319-46520-3_8.

10 Rüdiger Ehlers. Minimising deterministic Büchi automata precisely using SAT solving. In O. Strichman and S. Szeider, editors, *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT'10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 326–332. Springer, 2010.

11 Rüdiger Ehlers and Bernd Finkbeiner. On the virtue of patience: Minimizing Büchi automata. In Jaco van de Pol and Michael Weber, editors, *Model Checking Software – 17th International SPIN Workshop, Enschede, The Netherlands, September 27-29, 2010. Proceedings*, volume 6349 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2010. doi:10.1007/978-3-642-16164-3_10.

12 E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, June 1987.

13 Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In Hana Chockler and Georg Weissenbacher, editors, *Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18)*, volume 10981 of *Lecture Notes in Computer Science*, pages 567–577. Springer, 2018.

14 Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis – 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550. Springer, 2018. doi:10.1007/978-3-030-01090-4_34.

15 Juraj Major, František Blahoudek, Jan Strejček, Miriama Sasaráková, and Tatiana Zbončáková. ltl3tela: LTL to small deterministic or nondeterministic Emerson-Lei automata. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis – 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 357–365. Springer, 2019. doi:10.1007/978-3-030-31784-3_21.

16 David Müller and Salomon Sickert. LTL to deterministic Emerson-Lei automata. In Patricia Bouyer, Andrea Orlandini, and Pierluigi San Pietro, editors, *Proceedings of the Eighth International Symposium on Games, Automata, Logics and Formal Verification (GandALF'17)*, volume 256 of *EPTCS*, pages 180–194, September 2017.

17 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October – 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

18 Florian Renkin, Alexandre Duret-Lutz, and Adrien Pommellet. Practical “paritizing” of Emerson-Lei automata. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis – 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2020. doi:10.1007/978-3-030-59152-6_7.

Validation of QBF Encodings with Winning Strategies

Irfansha Shaik
Aarhus University, Denmark

Maximilian Heisinger
Johannes Kepler Universität Linz, Austria

Martina Seidl
Johannes Kepler Universität Linz, Austria

Jaco van de Pol
Aarhus University, Denmark

Abstract

When using a QBF solver for solving application problems encoded to quantified Boolean formulas (QBFs), mainly two things can potentially go wrong: (1) the solver could be buggy and return a wrong result or (2) the encoding could be incorrect. To ensure the correctness of solvers, sophisticated fuzzing and testing techniques have been presented. To ultimately trust a solving result, solvers have to provide a proof certificate that can be independently checked. Much less attention, however, has been paid to the question how to ensure the correctness of encodings.

The validation of QBF encodings is particularly challenging because of the variable dependencies introduced by the quantifiers. In contrast to SAT, the solution of a true QBF is not simply a variable assignment, but a winning strategy. For each existential variable x, a winning strategy provides a function that defines how to set x based on the values of the universal variables that precede x in the quantifier prefix. Winning strategies for false formulas are defined dually.

In this paper, we provide a tool for validating encodings using winning strategies and interactive game play with a QBF solver. As the representation of winning strategies can get huge, we also introduce validation based on partial winning strategies. Finally, we employ winning strategies for testing if two different encodings of one problem have the same solutions.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Theory of computation → Logic and verification

Keywords and phrases QBF, Validation, Winning Strategy, Equivalence, Certificates

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.24

Supplementary Material *Software*: `https://github.com/irfansha/SQval`

Funding *Maximilian Heisinger*: Supported by the LIT AI Lab funded by the State of Upper Austria.
Martina Seidl: Supported by the LIT AI Lab funded by the State of Upper Austria.

1 Introduction

Quantified Boolean formulas (QBFs) extend propositional formulas with universal and existential quantifiers over the Boolean variables [3], rendering their decision problem PSPACE-complete. As many application problems from artificial intelligence and formal verification have efficient QBF representations (see [16] for a survey) and as much progress has been made in the development of QBF solving tools [13], QBFs provide an appealing framework for solving such problems. In practice, however, obtaining correct and concise QBF encodings can be complex and error-prone as currently hardly any support for testing and debugging QBF encodings is available. The complexity of getting correct encodings comes on the one hand from the fact, that QBFs provide only a low-level language operating on the bit level

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 24; pp. 24:1–24:10
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and on the other hand from their interactive nature resulting from the quantifier alternations. The evaluation of a QBF can be seen as a two-player game between the existential and the universal player, where the existential player's goal is to satisfy the formula and the universal player's goal is to falsify the formula. Hence, models of true QBFs and counter-models of false QBFs are also called their *winning strategies*.

A winning strategy is a set of Boolean functions that defines how to set existential (universal) variables of a QBF to satisfy (falsify) a true (false) formula. A winning strategy provides the solution of the encoded application problem like the plan of a planning problem or the witness that no plan exists. Basically, winning strategies can be obtained in two forms: (1) statically, in terms of serialized functions that map existential variables to universal variables (or vice versa) [1], or (2) as interactive game play either with a QBF solver as opponent or by proof rewriting [6]. In contrast to SAT, where a solution is simply a variable assignment, the correctness of a winning strategy is challenging to validate, because of the dependencies between the variables. To prove that a winning strategy in serialized form is indeed a solution of a given QBF ϕ, a co-NP-hard problem has to be solved if ϕ is true, and an NP-hard problem has to be solved otherwise. This check can be automated by using a SAT solver. To show that a winning strategy is indeed a solution of an application problem, mainly remains a manual task. It is even non-trivial to find out if two variants of a problem encoding (e.g., a basic version and an optimization) share some common solutions.

We present a tool that supports the interactive testing of encodings based on serialized winning strategies in terms of Boolean functions as well as on dynamic validation, i.e., playing interactively against a QBF solver. We also propose a combination of both approaches for scalable validation based on partial winning strategies. To automate the testing process of an encoding, we implement a fuzz-testing approach for randomly exploring different parts of the search space. Fuzz testing has been successfully employed for testing solvers [5], but not for testing encodings. Finally, we present an approach to compare different variants of an encoding. While it is in general not feasible to prove that two encodings are equivalent, our tool supports testing if the two encodings share a common winning strategy. With a case study, we illustrate how our tool can be used to evaluate and understand QBF encodings.

2 Preliminaries

We consider closed QBF formulas in prenex normal form, i.e., of the form $Q_1X_1 \cdots Q_nX_n.\phi$, where quantifiers $Q_i \in \{\forall, \exists\}$, $Q_i \neq Q_{i+1}$, and X_i are disjoint sets of variables. The matrix ϕ is a propositional formula over $\bigcup X_i$. A QBF $\forall X\Pi.\phi$ is true iff both $\forall X'\Pi.\phi[x/\top]$ and $\forall X'\Pi.\phi[x/\bot]$ are true where $X' = X \setminus \{x\}$ and $\phi[x/t]$ is obtained from ϕ by replacing x by truth constant t. A QBF $\exists X\Pi.\phi$ is true iff $\exists X'\Pi.\phi[x/\top]$ or $\exists X'\Pi.\phi[x/\bot]$ is true. Often, the semantics of a QBF is also expressed as a two-player game: In the *ith* move, the values of the variables in X_i are chosen by the existential player if $Q_i = \exists$ and otherwise by the universal player. If all variables are assigned and the formula evaluates to true (false), then the existential (universal) player wins. A QBF is true (false) iff there is a winning strategy for the existential (universal) player. Winning strategies can be represented in terms of *Skolem/Herbrand functions*. A winning strategy for a true QBF $\Pi.\phi$ over existential variables X is a set $S = \{f_x \mid x \in X\}$ of Skolem functions such that each f_x is a Boolean function over the universal variables preceding x in the prefix and $\phi[X/S]$ is valid. Dually, a winning strategy for a false QBF $\Pi.\phi$ over universal variables Y is a set $H = \{f_y \mid y \in Y\}$ of Herbrand functions such that f_y is a Boolean function over the existential variables preceding y in the prefix and $\phi[Y/H]$ is unsatisfiable.

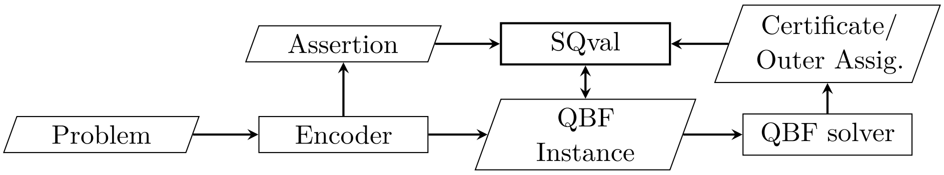


Figure 1 Validation workflow with SQval.

3 QBF Validation With Interactive Play

With our tool SQval (Scalable QBF Validator), we support the validation of QBF encodings independent of any specific QBF application. The general workflow is shown in Figure 1. The tool accepts both formulas in prenex conjunctive normal form (PCNF) and formulas in prenex non-CNF format. Hence, the QDIMACS format for PCNF formulas and the QCIR format [9] for prenex non-CNF formulas are supported.

3.1 Playing with Skolem/Herbrand Functions

Given a true QBF $\exists X_1 \forall Y_1 \exists X_2 \forall Y_2 \ldots \exists X_n \forall Y_n . \phi$ which has the set S of Skolem functions as one solution. These functions can be used to calculate the values of the existential variables in X_i based on provided values of the universal variables Y_j with $j < i$. As the variables of X_1 occur in the outermost quantifier block, they do not depend on any universal variables. Therefore, their Skolem functions are constant and can immediately be provided. Then the user has to enter the values for variables Y_1. Alternatively, they can also be randomly selected. Based on the values of the variables in Y_1, the values of the variables in X_2 are calculated. This procedure is repeated until all variables are assigned a value and the matrix ϕ evaluates to true. Evaluation of a false QBF works dually by using Herbrand functions.

There are solvers like Caqe [14] that construct Herbrand and Skolem functions during the solving process and there are frameworks that retrospectively extract functions from proofs produced by the solvers [11, 2, 4, 12]. Such functions allow to independently certify the correctness of a solving result by using a SAT solver. Little information, however, is provided on the correctness of the problem encoding. Here, we use the functions to "execute" test cases and interpret the results in an interactive manner. In our interactive play, Skolem functions automatically decide the moves of the existential player in the case of true instances. For false instances, Herbrand functions provide the moves of the universal player.

The Herbrand and Skolem functions have to be precomputed and can be provided in the AIGER format or as propositional formula in CNF format. In addition, encoding-specific assertions can be provided to SQval as CNF formulas. These assertions are checked under the full variable assignment at the end of the play. Such an assertion could be used, for example, to check if some goal condition is met or if some invariants are not violated in the game play. An example is given in the case study presented in Section 5.

The advantage of this method is the one-time cost of generating Skolem/Herbrand functions. Their evaluation is computationally cheap, because only truth values need to be propagated. However, even for simple problems, these functions can become very large and producing such functions often considerably slows down the solvers, because powerful pre- and inprocessing techniques have to be disabled. To overcome this drawback, we present an alternative approach in the following.

3.2 Playing with a QBF Solver

Given a true QBF of the form $\Phi_1 = \exists X_1 \forall Y_1 \Pi.\phi_1$ or a false QBF of the form $\Phi_2 = \forall Y_2 \exists X_2 \Pi.\phi_2$, most QBF solvers are able to provide assignments σ_{X_1} and σ_{Y_2} such that Φ_1 evaluates to true under σ_{X_1} and Φ_2 evaluates to false under σ_{Y_2}. If we apply σ_{X_1} on Φ_1, we obtain the true QBF $\Phi_1' = \forall Y_1.\phi'$. As this QBF has to be true for all assignments of variables Y_1, we pick now an interesting assignment and apply it on Φ_1' to obtain the true QBF Φ_1'' which starts with an existential quantifier block (or it has become the empty formula). Now we can ask a QBF solver for a satisfying assignment of these outermost variables and proceed in this way until all variables are finally assigned. Similarly, we can interactively evaluate Φ_2.

Based on this approach, we can replace large Skolem/Herbrand functions and avoid slower QBF solvers. The disadvantages of the approach is that a linear number of QBF problems must be solved: one for each round of validation.

3.3 Hybrid Validation

For many instances, the two interactive play approaches presented above are either limited by the size of the Skolem/Herbrand functions or by the costs of the QBF solver calls. As a solution, we suggest combining both approaches. First, we precompute Skolem/Herbrand functions only for the variables in the first k quantifier blocks of a QBF Φ. Based on these functions, we calculate the truth values of the variables which they define, and replace them respectively in Φ. Now we obtain a QBF Φ' with k quantifier alternations less and fewer variables. Then we proceed with evaluating Φ' by playing against a QBF solver.

The certification framework QBFcert [11] supports the generation of partial winning strategies, so we can use the respective options to obtain the functions of the variables from the first k quantifier blocks. We also provide an extractor for obtaining partial winning strategies from a full winning strategy by specifying the variables that should be considered. The partial winning strategies remain smaller than full winning strategies and we can take advantage of faster non-certifying solvers for the validation. Note that, one can generate assignments in the outer-most quantifier block with almost any non-certifying solver. In case the certifying solvers are too slow, one would use such an assignment to speed up the validation. In Section 5 we will demonstrate the memory/time tradeoff with partial strategies.

4 Common Winning Strategies of Two Encodings

Often, the same problem can be encoded in many ways resulting in formulas with different winning strategies. Consider for example two encodings of a two-player game for which we have a basic reference encoding that is very hard to solve, but which is most likely correct. Further, there is an optimized encoding which can be solved faster and which only has winning strategies that must also be solutions to the basic encoding. Therefore, we call the basic encoding *more relaxed* than the optimized encoding. To increase the trust in the optimized encoding, we want to validate if a found winning strategy is indeed a solution of the basic encoding. To this end, we want to take a winning strategy of one formula and enrich the other formula with this encoding. If this enriched formula has the same truth value, then we can conclude that the winning strategy of the first encoding is also a winning strategy of the second encoding. As two encodings might be defined over different variables, we need to introduce a set of common variables $\mathcal{C}$ that occur in both formulas (in practice some renaming of the variables not occurring in $\mathcal{C}$ might be necessary to avoid name clashes). This validation approach also works both for true and false formulas.

We first define a subsumption check between two true QBF formulas ϕ_1, ϕ_2. For this purpose, we use certificates, which are essentially winning strategies.

▶ **Definition 1.** *Let ϕ_1 and ϕ_2 be two true QBFs with common variables $\mathcal{C}$. We define ϕ_1 solution-subsumes ϕ_2 (written as $\phi_1 \sqsubseteq \phi_2$) iff for all winning strategies S for ϕ_1, also $\phi_2[S/\mathcal{C}]$ is true.*

We can only test $\phi_1 \sqsubseteq \phi_2$ for particular instances of S. To show the subsumption of ϕ_1 to ϕ_2, we need to show that a given strategy S of ϕ_1 is also a strategy for ϕ_2. Recall that a strategy is simply a function from universal variables to existential variables. As long as the same function works for ϕ_2, the subsumption relation is not refuted.

To show that, we first rewrite S to S' to avoid common name conflicts with ϕ_2. While rewriting S to S', we leave the common variables untouched (which are part of the winning strategy we consider). For example, in the game instances in the case study 5, we do not rewrite black player and white player moves. Finally, we create a new formula $S' \wedge \phi_2$. We claim that the new formula is True iff S' is a strategy for ϕ_2. Otherwise, checking this single S is sufficient to refute solution-subsumption, which is useful for bug detection. The S' formula essentially forces the existential variables given values to the universal variables. Since the QBF solver has to satisfy both S' and ϕ_2 which share common variables, the assignments to the common variables are always the same. In our game instances, the black moves for the opponent's white moves are forced by the rewritten strategy S'.

While for true formulas, the (possibly modified) winning strategy is just conjunctively added to the matrix, for false formulas it is additionally necessary to change the quantifier type of the variables defined by the winning strategy to existential. If the formula remains false, then the winning strategy of the first formula is also a winning strategy of the second formula. Since checking for common solutions has similar memory problems as the winning strategies for the interactive play presented in Section 3, we also support checking solution-subsumption with partial winning strategies. Our tool is completely agnostic to the completeness of a winning strategy. Examples are shown in the next section.

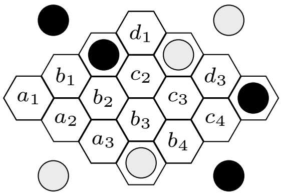
5 Case Study

We provide an open source implementation of validation and winning strategy equivalence. All benchmarks and data are available online.[1] To obtain a winning strategy, our tool SQval (Scalable QBF Validator) first generates a proof trace in QRP-format using the solver DepQBF [10]. Then it extracts the winning strategy using the QRPcert framework [11]. For interactive plays, it uses solver DepQBF for QDIMACS instances and solver Quabs [8] for QCIR instances. All computations for the experiments are run on a cluster.[2]

As a case study, we conducted an experiment with two QBF encodings for positional games. In particular, we compare two encodings for the game Hex: Lifted Neighbour-Based (LN) and Stateless Neighbour-based (SN) in [15]. In the Hex game, players (black, white) take turns to occupy empty positions on a NxN board with hexagonal cells. The player who connects appropriate opposite borders with a path of pegs of their own color wins the game. To demonstrate validation and equivalence checking, we first consider a small Hex

[1] `https://github.com/irfansha/SQval`

[2] `http://www.cscaa.dk/grendel-s`, each problem uses one core on a Huawei FusionServer Pro V1288H V5 server, with 384 GB main memory and 48 cores of 3.0 GHz (Intel Xeon Gold 6248R).



(a) Hein puzzle 12, before Hex preprocessing.

```
e 1 2 3 0
a 4 5 6 0
e 7 8 9 0
a 10 11 12 0
e 13 14 15 0
a 16 17 18 0
e 19 20 21 0
```

(b) Move variables in all Hein-12 QBF instances.

Table 1 Number of valid runs/ Number of invalid runs for each instance and each assertion.

	static			dynamic			hybrid		
Inst: / Assert:	GA	LPA	LBA	GA	LPA	LBA	GA	LPA	LBA
LN-Hein-12	100/0	100/0	100/0	100/0	100/0	100/0	100/0	100/0	100/0
SN-Hein-12	100/0	100/0	100/0	100/0	100/0	100/0	100/0	100/0	100/0
SN-R-Hein-12	100/0	100/0	91/9	100/0	100/0	84/16	100/0	100/0	84/16

instance, Hein-12 (see Figure 2a), due to Piet Hein [7]. We first preprocess the instance as in [15], resulting in 8 open positions. Using the preprocessed instance, we generate three QBF instances for a winning strategy of depth 7:

- LN-Hein-12 : Instance generated with LN, both players can only occupy empty positions.
- SN-Hein-12 : Instance generated with SN, black player can only occupy empty positions.
- SN-R-Hein-12 : Instance generated with SN with relaxed constraints for optimization, in which the black player is allowed to occupy previous black positions.

All three QBF instances solve the Hex instance, and their first 7 layers (cf. 2b) correspond to the moves taken, i.e., variables encoding 1 out of max 8 open positions (3 bits) per layer.

5.1 Validating Hein-12 Instances

The Hein-12 puzzle indeed has a winning strategy at depth 7, so all 3 instances are true formulas. For validation, we propose three assertions that are relevant to these encodings.

- Goal-Assertion (GA): "Goal is reached at the end of the play". For the LN instance, satisfying the goal constraint is specified by the assertion clause "`314 0`".
- Legal-Play-Assertion (LPA): "Black does not play on white positions". We generate inequality constraints between black moves and preceding white moves as a CNF.
- Legal-Black-Assertion (LBA): "Black does not play on black positions". We generate inequality constraints between different black moves as a CNF.

GA and LPA should hold for all instances, whereas LBA should only hold for the LN-Hein-12 and SN-Hein-12 instances. We use all 3 types of QBF validation for checking these 3 assertions on all 3 encodings. We run 100 iterations with a random generator with seeds ranging from 0-99. For hybrid validation, we use partial certificates up to depth 3 and validate the rest with a QBF solver. In Table 1, we present for each case the number of passing/failing runs. Indeed, both static, dynamic, and hybrid validation show some runs revealing the failure of assertion LBA for SN-R-hein-12, while all other tests pass. For Hein-12, all three validation techniques take a few seconds and a few MB for each iteration.

Table 2 Listing the result of the subsumption tests between instances for Q1 $\sqsubseteq$ Q2.

Q1: / Q2:	LN-Hein-12	SN-Hein-12	SN-R-Hein-12
LN-Hein-12	T	T	T
SN-Hein-12	T	T	T
SN-R-Hein-12	F	F	T

5.2 Equivalence Check for Hein-12

Validating GA and LPA increases the confidence in the correctness of the previous encodings. Additionally, we expect that LN-Hein-12 and SN-Hein-12 have the same winning strategies, since in both encodings, the black player can only play on open positions, and the same moves lead to equivalent states. This cannot be checked with our testing or fuzzing approach. Instead, we apply our subsumption check on all combinations of the 3 encodings. Table 2 shows the results from our subsumption check. Indeed, LN-Hein-12 and SN-Hein-12 appear to be equivalent (on the winning strategy returned by the solver). However, we found a strategy for SN-R that is not valid for SN and LN. Indeed, SN-R can play on already occupied black positions, which leads to invalid strategies for the LN and SN encodings. On the other hand, every move played in LN or SN is also valid in SN-R, resulting in subsumption in the other direction. These results are consistent with the intention behind the encodings.

5.3 Validating Larger Hex Instances With Partial Certificates

The Hein-12 instance has only 8 open positions after preprocessing, and we checked only for depth 7. On this small example, all 3 validation approaches worked equally well. Since the certificates remain small, the subsumption checks can be done within a few seconds. However, certificates grow exponentially in size with the number of variables in each layer and alternation depth. To show the difference between the validation strategies, we experiment with a harder Hex instance, Hein-09, which has 10 open positions after preprocessing, and we generate instances with a winning strategy of depth 9.

In Table 3, we observe that the full certificates in AAG format (Ascii And-Inverter Graph) are quite large, in the range of 532.7 MB – 7.8 GB. Note that the partial certificates are much smaller, but increasing with the level. Table 4 shows the resources required for generating the QRP traces for different settings, and for extracting certificates from the traces.

To show the difference between the 3 validation approaches on the harder instance Hein-09, we consider assertion GA for validation. From Table 5, we observe that static validation with full certificates for LN is infeasible. While we can validate the other encodings using full certificates, it takes up to 20 GB. Dynamic validation performs well on SN and SN-R, while it takes 1203 seconds for validating a single iteration of an LN instance. Note that

Table 3 Size in Bytes of (partial) certificates in AAG format, for each encoding instance with increasing levels of partial certificates. QRP trace is the size of the trace generated by solver DepQBF.

Enc: / Cert:	Full	L1	L3	L5	L7	L9	QRP trace
LN-Hein-09	7.8G	108	1.1K	21.8K	434K	8.2M	6.8G
SN-Hein-09	641.6M	96	1.1K	23.4K	437.3K	8.2M	344M
SN-R-Hein-09	532.7M	96	1.2K	24.2K	461.1K	8.7M	394.5M

Table 4 Time and Memory used for generating the QRP trace and extracting certificates from it.

	Peak Memory (MB)				Time Taken (Sec)			
Enc: / Cert:	QRP trace	Full	L3	L9	QRP trace	Full	L3	L9
LN-Hein-09	13470	10390	7580	9750	1165	261	59	233
SN-Hein-09	615.46	1.54	1.54	1.54	54	20	21	15
SN-R-Hein-09	535.9	1.54	1.54	1.54	54	13	4	15

Table 5 Peak Memory (PM) in MB and Time Taken (TT) in seconds for assertion GA (seed 0).

	static		dynamic		hybrid-L3	
Inst:	PM	TT	PM	TT	PM	TT
LN-Hein-09	–	TO	64.1	1203	**1.54**	**1.6**
SN-Hein-09	20.08K	2100	1.54	9.4	**1.53**	**0.4**
SN-R-Hein-09	18.35K	1226	**1.53**	5.4	**1.53**	**0.4**

to run 100 iterations with different seeds, we would need approximately 100*1203 seconds. Hybrid validation performs clearly the best in both time and memory, never exceeding a couple of seconds or 2 MB. Of course, generating a partial certificate via QRP trace is still the bottleneck for hybrid validation. However, we only pay a one-time cost for generating partial certificates, which can be used any number of times for validation or subsumption checks.

We will now experiment with subsumption checking, using full or partial certificates. For Hein-09 the QBF instances appended with a certificate can exceed 15 GB in CNF. Note that, we only need partial certificates with all existential black move variables, i.e., L9 in Table 3. These never exceed 10 MB (in AAG format), so a complete subsumption check with L9 partial certificates only is feasible. We compute subsumption checks on all combinations of 3 encodings, i.e., for each combination we append one QBF with the certificate of the other, and use a QBF solver. From Table 6, it is clear that subsumption with full certificates blows up, often exceeding 50 GB of memory. In fact, we could not generate the appended instance with LN certificates since the instances themselves can exceed 20 GB.

On the other hand, with L9 certificates, we could check non-subsumption for SN-R with SN and LN, taking 446 and 84 seconds, respectively. DepQBF runs out of time when trying to prove the subsumption cases, but never uses more than 2 GB for solving with L9 certificates. Intuitively, proving non-subsumption is indeed easier than proving subsumption. One could try to use preprocessors, to speed up the subsumption checks for L9 certificates, but full certificate subsumption would still be out of reach.

Table 6 Peak Memory in GB during subsumption checks between Hein-09 instances.

	LN-Hein-09		SN-Hein-09		SN-R-Hein-09	
Inst:	Full	L9	Full	L9	Full	L9
LN-Hein-09	-	1.53	-	1.57	-	1.57
SN-Hein-09	63.7	1.53	63.7	1.6	63.7	1.6
SN-R-Hein-09	50.74	1.6	50.78	1.6	50.8	1.69

6 Conclusion and Future Work

In this paper, we proposed validation techniques with winning strategies and interactive play with a QBF solver. For scalable validation, we proposed using partial winning strategies for outer layers and interactive play for deeper layers. We extended the idea of validation to solution-equivalence of encodings that have some common winning strategy. To evaluate various techniques proposed, we conducted a case study on 2-player game encodings for the game Hex. We showed that with the use of winning strategies, one can increase the confidence of encoding correctness. In the most scalable approach, generating QRP traces remains the bottleneck, as solvers generate complete traces. While checking solver correctness requires complete traces, partial traces/certificates are sufficient for encoding validation. One future research direction would be to allow QBF solvers to generate partial traces efficiently.

References

1 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods Syst. Des.*, 41(1):45–65, 2012. doi:10.1007/s10703-012-0152-6.

2 Marco Benedetti. Extracting certificates from quantified boolean formulas. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 47–53. Professional Book Center, 2005. URL: http://ijcai.org/Proceedings/05/Papers/0985.pdf.

3 Olaf Beyersdorff, Mikolás Janota, Florian Lonsing, and Martina Seidl. Quantified Boolean Formulas. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1177–1221. IOS Press, 2021.

4 Roderick Bloem, Vedad Hadzic, Ankit Shukla, and Martina Seidl. Ferpmodels: A certification framework for expansion-based qbf solving. In *Proc. of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) 2022*, September 2022.

5 Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proc. of the 13th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2010)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2010.

6 Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 546–553. IJCAI/AAAI, 2011.

7 Ryan B. Hayward and Bjarne Toft. *Hex, the full story.* AK Peters/CRC Press/Taylor Francis, 2019.

8 Jesko Hecking-Harbusch and Leander Tentrup. Solving QBF by abstraction. In *Proc. of the 9th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF)*, volume 277 of *EPTCS*, pages 88–102, 2018. doi:10.4204/EPTCS.277.7.

9 Charles Jordan, Will Klieber, and Martina Seidl. Non-CNF QBF solving with QCIR. In *AAAI-16 Workshop on Beyond NP*, February 2016.

10 Florian Lonsing and Uwe Egly. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In *CADE*, volume 10395 of *Lecture Notes in Computer Science*, pages 371–384. Springer, 2017. doi:10.1007/978-3-319-63046-5_23.

11 Aina Niemetz, Mathias Preiner, Florian Lonsing, Martina Seidl, and Armin Biere. Resolution-based certificate extraction for QBF. In *Proc. of the 15th Int. Conf. on the Theory and Applications of Satisfiability Testing (SAT)*, volume 7317 of *Lecture Notes in Computer Science*, pages 430–435. Springer, 2012. doi:10.1007/978-3-642-31612-8_33.

12 Tomás Peitl, Friedrich Slivovsky, and Stefan Szeider. Polynomial-time validation of QCDCL certificates. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proc. of the 21st Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2018)*, volume 10929 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2018. doi:10.1007/978-3-319-94144-8_16.

13 Luca Pulina and Martina Seidl. The 2016 and 2017 QBF solvers evaluations (qbfeval'16 and qbfeval'17). *Artif. Intell.*, 274:224–248, 2019. doi:10.1016/j.artint.2019.04.002.

14 Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In *FMCAD*, pages 136–143. IEEE, 2015. URL: https://www.react.uni-saarland.de/publications/RT15.pdf.

15 Irfansha Shaik, Valentin Mayer-Eichberger, Jaco van de Pol, and Abdallah Saffidine. Implicit state and goals in QBF encodings for positional games (extended version). *CoRR*, abs/2301.07345, 2023. doi:10.48550/arXiv.2301.07345.

16 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A Survey on Applications of Quantified Boolean Formulas. In *Proc. of the 31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 78–84. IEEE, 2019.

Combining Cubic Dynamical Solvers with Make/Break Heuristics to Solve SAT

Anshujit Sharma
Department of Electrical and Computer Engineering, University of Rochester, NY, USA

Matthew Burns
Department of Electrical and Computer Engineering, University of Rochester, NY, USA

Michael C. Huang
Department of Electrical and Computer Engineering, University of Rochester, NY, USA

Abstract

Dynamical solvers for combinatorial optimization are usually based on 2^{nd} degree polynomial interactions, such as the Ising model. These exhibit high success for problems that map naturally to their formulation. However, SAT requires higher degree of interactions. As such, these quadratic dynamical solvers (QDS) have shown poor solution quality due to excessive auxiliary variables and the resulting increase in search-space complexity. Thus recently, a series of cubic dynamical solver (CDS) models have been proposed for SAT and other problems. We show that such problem-agnostic CDS models still perform poorly on moderate to large problems, thus motivating the need to utilize SAT-specific heuristics. With this insight, our contributions can be summarized into three points. First, we demonstrate that existing *make*-only heuristics perform poorly on scale-free, industrial-like problems when integrated into CDS. This motivates us to utilize *break* counts as well. Second, we derive a relationship between *make*/*break* and the CDS formulation to efficiently recover *break* counts. Finally, we utilize this relationship to propose a new *make*/*break* heuristic and combine it with a state-of-the-art CDS which is projected to solve SAT problems several orders of magnitude faster than existing software solvers.

2012 ACM Subject Classification Mathematics of computing → Probabilistic algorithms; Computer systems organization → Analog computers

Keywords and phrases Satisfiability, Ising machines, Stochastic Heuristics, Natural Computation

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.25

Supplementary Material *InteractiveResource (Experimental data, benchmarks and solver infrastructure)*: https://ising.ece.rochester.edu/sat

1 Introduction

The slowdown of general purpose computing has given rise to novel architectures to solve NP-Hard problems. One such approach is to go beyond the von Neumann paradigm and leverage systems whose evolution under physical laws carries out certain type of computation efficiently. This approach has shown potential for success at least in combinatorial optimization problems. In this regard, most of the literature has focused on quantum computing: specifically on Quantum Annealing (QA) [12, 21] and Adiabatic Quantum Computing (AQC) [2]. Recently, another non-von Neumann approach: Ising machines, has been gaining traction. The state-of-the-art Ising machines work completely in the classical regime relying on extremely fast dynamics of the system. Hence, these are less sensitive to noise when compared to quantum computers. Some notable examples are using coupled oscillators [52], capacitors in a resistive network [1, 46, 57], and modulated pulses of light [31].

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 25; pp. 25:1–25:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

These approaches have shown extraordinary performance on the weighted MaxCut problem when compared to software based approaches [1,27,46]. However, applications to SAT have been less successful. This is due to ① the lack of support for super-quadratic interactions; and ② the failure to leverage problem-specific information.

In this work, we examine and alleviate these shortcomings so as to revive the fast solution-finding capabilities of Ising machines. We focus on the Ising model of computation, both due to its successful implementation as fast dynamical hardware accelerators [1,31,32,52] and as algorithms [27]. We limit ourselves to 3-SAT due to problem reducibility and simplicity of discussion. We will specifically base our analysis using simulations of an Ising machine proposed recently [57], as it represents a near-term achievable piece of hardware.

The novel contributions of this work can be summarized into three points:

1. Demonstrating the shortcomings of previous super-quadratic solvers and heuristic proposals on uniform random and scale-free problems.
2. Deriving a relation between cubic dynamical formulations and the *make*/*break* counts of variables in a 3-SAT formula.
3. Proposing novel *make*/*break* heuristics by leveraging the cubic formulation and demonstrating their viability by comparing a simulated dynamics-based solver against state-of-the-art software SAT solvers.

The rest of the paper is organized as follows. Section 2 provides a background on the Ising model and related work. Section 3 introduces a cubic formulation for 3-SAT and demonstrates the shortcomings of problem-agnostic dynamical solvers. Building on this insight, Section 4 demonstrates and analyzes the inadequacy of existing *make*-only heuristics in solving scale-free problems. This motivates us to also utilize *break* counts. We then derive a relationship that enable us to easily recover *break* counts from the cubic formulation itself. We utilize this relationship to propose a new heuristic using both *makes* and *breaks*. In Section 5, we combine this heuristic with a state-of-the-art cubic dynamical system and compare it against existing CDCL and SLS solvers. Finally, we conclude our findings in Section 6 as well as propose future directions for research.

2 Background

2.1 Preliminaries

When discussing SAT formulas, we use the notation introduced in “The Handbook of Satisfiability” [13]. Unless otherwise specified, all problems discussed are in 3-SAT form. N and M refer to the number of variables and clauses respectively. $x_n \in \{0,1\}$ is used to denote an arbitrary variable, and $\mathsf{a} \in \{0,1\}^N$ is the full assignment vector. Uniform and scale-free problems used for testing are generated using the methodology described in a work by Ansótegui et al. [4,5].

2.2 Quadratic Models: Ising and QUBO

The Ising model was originally formulated by Wilhelm Lenz and solved in a simplified form by his student Ernst Ising [38]. It describes a system of magnetic *spins* (s_i), expressed in one dimension. Each spin takes the values, $s_i = \pm 1$. Each pair of spins (s_i, s_j) is “coupled” with some coefficient J_{ij}. An external field h_i can also exist, which imposes some linear coefficient to the spins. The overall energy of an N-spin system is expressed via the *Hamiltonian* H:

$$H(s, J, h) = -\sum_{i<j}^{N} J_{ij} s_i s_j - \sum_{i}^{N} h_i s_i = -s^T J s - h^T s \tag{1}$$

An Ising system will seek a state s such that $H(s)$ is minimized. Hence, $J_{ij} > 0$ implies spin affinity, known as *ferromagnetism*: s_i will tend to equal s_j. $J_{ij} < 0$ implies spin repulsion, known as *antiferromagnetism*: s_i will tend to equal $-s_j$.

When referring to Ising formulation parameters, s describes the complete spin state vector, J the complete coupling matrix. Subscripts will be added to indicate a single element, for instance s_i or J_{ij}. A state vector s^* for which $H(s^*) = 0$ is referred to as a *ground state* of H. We can express any given state vector for a SAT encoded Hamiltonian as an assignment to the original CNF problem, where $s_i = 1 \rightarrow x_i$, and $s_i = -1 \rightarrow \bar{x}_i$.

Another equivalent formulation is the Quadratic Unconstrained Binary Optimization (QUBO) form, where variables x_i take values in $\{0, 1\}$. An Ising formula can be trivially transformed into a QUBO formula using the following replacement rule for spins:

$$s = 2x - 1 \tag{2}$$

Formulating SAT as a QUBO problem is cleaner than its equivalent Ising formulation. Hence, in our discussions, we will use QUBO formulas, where $x_i = 1$ indicates "true" and $x_i = 0$ indicates "false", and $\mathtt{a} \in \{0, 1\}^N$ denotes the variable assignment vector. For convenience, we will refer to a dynamics-based QUBO/Ising solver as a *quadratic dynamical solver* (QDS). A QDS can be implemented in a wide variety of mediums [31, 32, 52], but we will focus on a CMOS-compatible hardware proposed by Afoakwa et al. [1] called Bistable Resistively-coupled Ising Machine (BRIM). There are some compelling reasons to use BRIM as the baseline for comparison:

1. *CMOS compatibility*: Unlike quantum systems and many other Ising machines, BRIM is electronic-based and can be fabricated using today's CMOS technology. This allows for easier extension of its design and integration with other heuristics to improve its performance. The proposed design is also more feasible and energy efficient in the near term as discussed in previous work [1].
2. *All-to-all connectivity*: Many Ising machine implementations have limited connectivity between spins which greatly limits its true capacity. To solve problems on such machines, one needs to transform the input graph into another (much bigger) graph that the hardware can map; a process called *embedding*. Embedding is NP-Hard in itself [20, 51]. BRIM supports all-to-all connectivity and thus, doesn't suffer from this problem.
3. *Extremely fast dynamics*: Unlike variants of *Coherent Ising Machines* (CIM) [31] which rely on FPGA computation to *emulate* coupling, the time evolution of BRIM is completely done naturally based on physics. Thus, BRIM can achieve good solutions very quickly as is established in previous works [1, 46].

Specifics on the BRIM model used for simulation is explained later in Section 4.3.1 and the pseudocode can be found in Appendix A. Throughout this work, we assume a variant of BRIM model with quantized nodal interactions [57].

2.3 Related Work

Physics-Based Optimization for SAT

Optimization literature has previously utilized physical computational methods. Myriad dynamical systems have been proposed which implement the quadratic Ising model to solve NP-Hard optimization problems, including time-evolving quantum systems [2, 24, 32], modulated optical pulses [31], coupled electronic oscillators [52], and resistively coupled capacitors [1]. These approaches have shown success in natively quadratic problems such as graph MaxCut, however their application to SAT has been largely unsuccessful as we will see. As an example, in one work [26], the authors proposed to optimize the Maximal Independent Set (MIS) reduction of 3-SAT with quantum annealing. In another work [15], the authors demonstrated the theoretical feasibility of gate-based quantum computing using noise-free QAOA simulations. In the near-term, quantum computers suffer from noise-induced errors which severely limits its performance and scalability [7, 11, 47, 53].

Optimization software algorithms loosely inspired by physical dynamics have also been proposed. The best known is simulated annealing (SA) [35]. SA minimizes a given cost function by taking inspiration from metallurgical processes with gradually decreasing temperature. Other notable physics-inspired examples are evolutionary algorithms [6].

Algorithms which directly simulate physical phenomena for optimization have also been proposed. Among these, simulated bifurcation (SB) [27] and continuous-time dynamical system (CTDS) [23] are primary examples. The former simulates chaotically bifurcating Ising models, the latter a chaotic system with exponentially growing factors. A GPU implementation of CTDS was shown to outperform MiniSAT in certain large problems [40].

Hardware SAT Solvers

Hardware acceleration of SAT algorithms broadly falls into two categories: total solvers (implementing an algorithm in its entirety) and subset accelerators (implementing specific operations of a SAT algorithm). The former generally implement SLS algorithms [30, 41, 48] due to their simpler heuristics. In one work, the authors proposed a novel combination of naturally stochastic "p-bit" hardware to accelerate a simplified variant of ProbSAT [48] (see Section 4.1). There are examples of total CDCL hardware [28], however complex heuristics preclude efficient implementation. Subset solvers are commonly Boolean constraint propagation (BCP) accelerators [22]. BCP is particularly computationally demanding in CDCL (upwards of 90% of computation time [22, 50]). The fixed proportion means BCP accelerators can *only* provide an 8-10× speedup by Amdahl's law. Other novel solvers include an analog circuit proposed [56] and demonstrated [17] by implementing the aforementioned CTDS algorithm [23]. Interested readers are referred to an in-depth survey for a more thorough treatment of SAT hardware acceleration [50].

High-degree Dynamical Solvers

The inability of classical quadratic models to generalize to higher-degree polynomial interactions has motivated a recent surge in Ising-like high-degree models[1]. Formulations of k-SAT and NAE-SAT have been proposed and simulated in various dynamical systems [9, 16]. The SB algorithm has been extended to support high-degree polynomial interactions [33] with potential applications to highly parallel SAT solvers. These models have been shown to significantly outperform the traditional quadratic reductions described in literature [36, 38, 44, 51]. However, as we will discuss in Section 3.2, models implementing problem-agnostic local minima escaping heuristics demonstrate poor solution quality for moderate to large problems.

[1] Note that, all the citations refer to high-degree polynomial interactions as "high-order", not to be confused with *high-order logic*.

3 High-degree Polynomial Formulation

3.1 Reformulating SAT

Finding the globally minimum energy of a QDS model is proven to be NP-Hard, allowing for a large number of problems to be reformulated as QDS Hamiltonians [38]. An initial proposal to reduce 3-SAT to a QUBO model was to formulate it as a "Maximal Independent Subset" (MIS) problem [38]. While MIS can be more naturally encoded into the QUBO model, encoding an M clause CNF requires $3M$ QUBO spins, an unacceptable level of search-space bloat. Instead, we propose a cubic Hamiltonian H_C as a "natural" expression of 3-SAT. Thus, for a given M-clause CNF 3-SAT formula:

$$f = \bigwedge_{i=1}^{M} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,2}) \tag{3}$$

the proposed Hamiltonian H_C is given by:

$$H_C = \sum_{i=1}^{M} g(\ell_{i,1})g(\ell_{i,2})g(\ell_{i,3}) \tag{4}$$

where for each clause literal $\ell_{i,j}$

$$g(\ell_{i,j}) = \begin{cases} (1 - x_n) & \ell_{i,j} = x_n \\ x_n & \ell_{i,j} = \bar{x}_n \end{cases} \tag{5}$$

Here x_n is the variable corresponding to the j^{th} literal of the i^{th} clause, $n \in \{1, ..., N\}$. Each variable in a problem only has one associated spin, hence this formulation requires N spins to represent the complete CNF. k-SAT problems for $k > 3$ have a natural analogue with higher-degree terms, as discussed in a recent work [16]. However, we limit our focus to 3-SAT.

We observe that $g(\ell_{i,j}) \in \{0, 1\}$, with $g(\ell_{i,j}) = 0$ corresponding to true $\ell_{i,j}$ and $g(\ell_{i,j}) = 1$ corresponding to false $\ell_{i,j}$. A clause $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ is satisfied if and only if at least one of its literals is true. Therefore, we observe:

▶ **Proposition 1** (Clause Satisfaction). *A clause $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ is satisfied by the state vector* a *if and only if $g(\ell_{i,1})g(\ell_{i,2})g(\ell_{i,3}) = 0$.*

Since the Hamiltonian H_C is a sum over such clause-derived products, we conclude:

▶ **Lemma 2** (Ground States are SAT). a *is a ground state of H_C ($H_C(\mathsf{a}) = 0$) if and only if* a *satisfies the underlying CNF formula f.*

The logical equivalence between H_C and a given CNF formula makes H_C a desirable Hamiltonian for QDS implementation. However, the QDS model is limited to quadratic and linear terms only, and it lacks support for 3^{rd}-degree polynomial (cubic) interactions which is required in H_C. Therefore, to implement H_C in a QDS format, one must "quadratize" all cubic terms by introducing extra auxiliary spins and extra terms to penalize undesirable states [25, 36, 44][2].

[2] We do note that, the MAX-2-SAT reduction of 3-SAT [54] does map naturally to the QDS format. However, mathematically, it is equivalent to the one proposed by Kolmogorov et al. [36] and thus, suffer from the same problems.

In a recent work [16], the authors have demonstrated a drop in solution quality after such "quadratization". This motivated them to propose an Ising-formulated $\{+1, -1\}$ high-degree polynomial model. With experimental analysis for uniform random problems, the authors showed its superior performance compared to QDS.

Throughout our discussion, we limit our scope to solvers with support for cubic interactions and refer to these as *cubic dynamical solvers* (CDS).

3.2 Problem Agnostic Heuristics

As with naive gradient descent, dynamics-based solvers (QDS and CDS) can be trapped in local minima in the energy landscape. Classical optimization algorithms such as simulated annealing (SA) [35] overcome such local traps by introducing stochastic behavior (occasionally accept a "bad" state). It has been proven that given enough time, SA can converge to the global optimum [39]. Thus, a common approach for hardware solvers is to mimic the behavior of the SA algorithm by introducing random noise to perturb the system. Example noise sources include varying external oscillations [52], random polarity flips [1], and transverse quantum field fluctuations [32]. The strength of these perturbations decrease over the course of the run according to an *annealing schedule*. The primary motivation of such a schedule is to broadly explore the search-space initially and then (hopefully) settle into some global optimum towards the end. In general, these perturbations are applied randomly across problem variables, taking into account no knowledge of problem type or structure.

A recently proposed CDS used such problem agnostic perturbations [16], as have previous works proposing dynamics-based solvers [1, 26]. Here, we refer to this as the `Anneal` heuristic. Just like prior works on BRIM [1, 46], we use a linear annealing schedule to induce artificial spin flips. Let p_0 and p_1 be the start and end probabilities respectively. Then, at any given time t and a total run time of T, the instantaneous probability p to flip a spin is given by: $p = p_0 - \frac{t(p_0 - p_1)}{T}$. Thus, each spin is treated as an i.i.d. Bernoulli random variable with flip probability p.

Fig. 1 shows the performance of CDS with `Anneal` heuristic on 5 sets of 100 generated uniform random problems with $M/N = 4.25$ and $N \in \{100, 200, 300, 400, 500\}$. The left figure shows the number of instances solved as N increases with a constant run time of 0.1 ms. The system easily finds solutions to 99% of the 100-variable problems, however the solved proportion drops quickly to 26% for $N = 500$. We also consider the possibility that the larger instances require longer system evolution times. Fig. 1 [right] shows the number of solved instances for the 500-variable set as run time is increased up to 2 ms (20× longer). We observe only a minor increase in solved proportion to 48%. Parameter sweeps of p_0/p_1 are also unable to improve solution quality for larger instances.

Theory suggests that all problems would be solved given a sufficiently long run time with a conservative, ergodicity-preserving schedule [39]. However, these results suggest that such schedules/anneal times would not represent a substantial speedup (if any) over other existing solutions. To give some context, WalkSAT is capable of solving the same 500 variable instances in as little as 5 ms. Therefore, we believe that such a problem-agnostic or "blind" annealing is insufficient.

4 Integrating SLS Heuristics

The reason for the comparatively poor performance of the problem-agnostic `Anneal` heuristic lies largely in the existence of this conference. SAT is a well-studied problem whose peculiarities are exploited by solver heuristics. CDCL-style clause learning uses Boolean

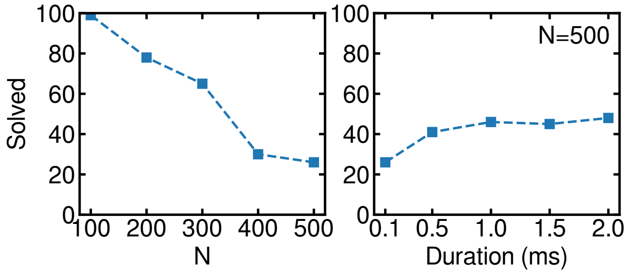


Figure 1 Evaluation of CDS with a problem-agnostic `Anneal` heuristic. Left: Number of uniform random instances solved out of 100 with a constant run time of 0.1 ms as problem size increases. Right: Number of 500 variable instances solved as we increase the system run time.

resolution to actively store information on the problem at hand, and VSIDs have been shown to exploit large scale problem structures [37]. SLS algorithms such as WalkSAT [34, 45] include a veritable buffet of different heuristics for variable selection based on various scoring and ranking mechanisms.

Problem-agnostic dynamical systems have yet to adopt such advanced heuristics to exploit the existing body of SAT-specific knowledge. Prior implementations of BRIM used the `Anneal` heuristic. While sufficient for more unstructured problems (e.g. MaxCut [1, 32]), such annealing schedule falls short for a structured problem like 3-SAT requires. The results shown in Fig. 1 demonstrate the shortcomings of this uniform random heuristic. This motivates us to propose ProbSAT-like stochastic heuristics to increase search-space efficiency of CDS. We first discuss why it is a good idea to start with the ProbSAT heuristic, then move onto discussion of a hardware friendly variant and further improvements.

4.1 The ProbSAT heuristic and its variant

ProbSAT [8] defines a probability distribution for all problem variables based upon an internally determined score. The score in turn depends on the variable's *make* and *break* counts. We use standard definitions for these terms, restated here for clarity.

▶ **Definition 3** (Make Clause). *For a given assignment* $\mathtt{a}$*, the* ***make*** *count of variable* x_n *is the number of* *newly satisfied* *clauses after flipping* x_n*. We denote it as* $\mathcal{M}_n(\mathtt{a})$*.*

▶ **Definition 4** (Break Clause). *For a given assignment* $\mathtt{a}$*, the* ***break*** *count of variable* x_n *is the number of* *newly unsatisfied* *clauses after flipping* x_n*. We denote it as* $\mathcal{B}_n(\mathtt{a})$*.*

We observe that the net change in SAT clauses by flipping variable x_n with current assignment $\mathtt{a}$ is $\mathcal{M}_n(\mathtt{a}) - \mathcal{B}_n(\mathtt{a})$. For simplicity, we omit the argument $\mathtt{a}$ from subsequent uses.

Note that for a particular variable x_n, any currently UNSAT clause in which it appears is necessarily a *make* clause, while *break* clauses are a subset of currently SAT clauses. The original ProbSAT paper examined two candidate distribution functions: polynomial and exponential. For both classes of function, the authors found that using $\mathcal{B}_n$ alone provided a more effective heuristic.

ProbSAT explores the search-space solely based upon the probability distribution, hence the logic is relatively simple compared to other SLS algorithms. Accordingly, a ProbSAT-style algorithm is suitable for hardware implementations [48, 49]. The original algorithm works sequentially, choosing random UNSAT clauses and sampling from its variables based on the defined distribution. However, sequential flips are both impractical and undesirable for CDS integration, as they require complicated synchronization and information propagation across the system. Instead, it is easier to allow variables to flip in parallel (distributed manner).

Building on this motivation, researchers proposed a *Biased Random Walk* (BRW) heuristic as a modification to ProbSAT [48]. The approach allows parallel flips and relies solely on $\mathcal{M}_n$. The authors proposed two different update rules for the flipping probability, depending on $\mathcal{M}_n$: *linear* and *nonlinear*. The *linear* rule probability, p_l is parameterized by p_{step}, and for a particular variable x_n, $p_l(x_n) = \mathcal{M}_n \times p_{step}$. The *nonlinear* rule has two parameters: p_{init} and p_{step}. The first *make* clause of a variable x_n sets $p_{nl}(x_n) = p_{init}$ and every subsequent *make* contributes p_{step}. Our simulations of both rules using uniform random problems verified that the *nonlinear* rule produced much better results. Hence, we use this rule whenever we refer to BRW[3].

4.2 Combining BRW with CDS

In the original paper [48], the authors implemented the BRW algorithm in hardware using stochastic Magnetic Tunnel Junction (MTJ) devices to represent the variables. It turns out, $\mathcal{M}_n$ are easy to calculate in such a parallel system. The authors proposed a digital circuit which couples the states of individual variables to determine clause state. Instead of designing a stand-alone BRW hardware, we propose a CDS system, cBRIM (BRIM with support for cubic terms), so as to leverage its fast dynamics (spins could flip every 20 ps [1]), along with a BRW-like heuristic as its annealing algorithm. We refer to this solver as `cBRIM-BRW`. The details of cBRIM are explained later in Section 4.3.1.

Experimental analyses demonstrated the superior performance of an MTJ-based BRW solver over WalkSAT and ProbSAT [48]. However, the experiments were only done with uniform random problems. To consider problems of a heterogeneous structure, we consider the variable interaction graph (VIG) of the formula. The problem variables form the VIG vertex set, with edges connecting variables co-appearing in a clause[4]. The VIG was originally introduced as a means of reasoning about CDCL resolution and algorithm behavior [43], but has since become a standard means of reasoning about problem structure [3]. It has been established that industrial problem VIGs exhibit *scale-free* structure [4, 5]. Specifically, the number of clauses each variable appears in (and consequently the vertex degree in the VIG) follows a power-law distribution. In such a model, the probability of any given node to have degree d has the form: $P(d) \propto d^{-\beta}$. Industrial instances generally have $\beta \in [2, 3]$ compared to $\beta > 18$ for uniform random problems. This implies that there is a rapid fall in the probability of high-degree variables for uniform random problems and hence relatively low degree variance ($< 0.3\times$ the average degree) compared to that of industrial ($> 1.5\times$) [4].

Consequently, real-world problems do not follow the Erdös-Renyi model used to generate the uniform random instances. Therefore, we simulate `cBRIM-BRW` and run custom generated power-law problems of 500 and 1000 variables with different values of β parameter [5]. For all the runs, we set $p_{init} = 0.03$, $p_{step} = 0.2$ for scale-free problems, and $p_{init} = 0.07$, $p_{step} = 0.9$ for uniform-like problems: both tuned for the best performance on 500 variable instances.

Fig. 2 shows how many problems `cBRIM-BRW` solved given different β values. We can observe that, as β increases (problems become more uniform-like), the performance of `cBRIM-BRW` improves generally. This is expected as BRW is demonstrated to work well

[3] p_l or p_{nl} can be greater than 1. The authors didn't elaborate, but presumably the probability is capped at 1.

[4] This differs slightly from the bipartite graph form described in another work [5], however the significance of vertex degree is equivalent.

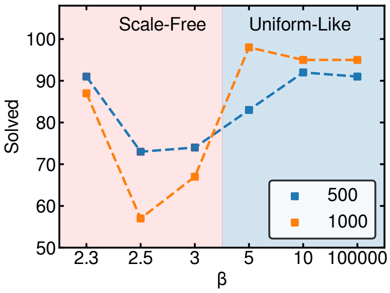


Figure 2 Number of problems solved by `cBRIM-BRW` out of 100 power-law distributed problems with 500 and 1000 variables for increasing β. We set cutoff run time of 0.11 ms and 1.1 ms for 500 and 1000 variables respectively.

for uniform random problems [48]. However, when $\beta \in [2.5, 3]$, the performance degrades. Moreover, the parameters tuned for 500 variable scale-free problems did not work well on larger problems: particularly in the case of $\beta = 2.5$. Thus, the optimal choice of parameters in `cBRIM-BRW` appears to be sensitive to the size of scale-free problems, far from ideal for practical use. These results lead us to conjecture that $\mathcal{M}_n$ alone may not contain sufficient information to determine advantageous flips. Therefore, we propose to leverage both $\mathcal{M}_n$ and $\mathcal{B}_n$ in a stochastic heuristic.

4.3 Towards a better heuristic

A primary consideration for algorithms like BRW to use only $\mathcal{M}_n$ is that $\mathcal{B}_n$ is much harder to obtain in hardware. It relies both on problem structure and current variable assignments, imposing more hardware complexity. However, it turns out, the CDS formulation does provide some information that we can leverage to get $\mathcal{B}_n$ without requiring complex hardware.

4.3.1 Recovering Break Counts

Here, we discuss how to leverage inherent information from the CDS formulation to recover $\mathcal{B}_n$. Let us denote H_{Cn} as the Hamiltonian of all the clauses involving the variable x_n:

$$H_{Cn} = \sum_{x_n \in C_i} g(\ell_{i,1})g(\ell_{i,2})g(\ell_{i,3}) \tag{6}$$

where we denote $\ell_{i,2}$ and $\ell_{i,3}$ as the non-x_n literals appearing in each clause and $\ell_{i,1} = \{x_n, \bar{x}_n\}$ without loss of generality. From H_{Cn}, we now consider only the terms containing x_n. Recall from the definition of $g(\ell_{i,j})$ in Eq. 5 that these terms will have the form $\pm x_n g(\ell_{i,2})g(\ell_{i,3})$.

▶ **Definition 5** (x_n-Relevant Form)**.** *A clause $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ where $\ell_{i,1} \in \{x_n, \bar{x}_n\}$ has x_n-relevant form $\mathcal{R}_{i|x_n} = \pm g(\ell_{i,2})g(\ell_{i,3})$*

We now define the state of a clause containing x_n, independent of x_n's assignment.

▶ **Definition 6** (x_n-UNSAT Clause)**.** *Let clause $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ where $\ell_{i,1} \in \{x_n, \bar{x}_n\}$. C_i is x_n-UNSAT iff the subclause $(\ell_{i,2} \vee \ell_{i,3})$ is UNSAT.*

Suppose C_i contains the literal x_n. Then $\mathcal{R}_{i|x_n} = -g(\ell_{i,2})g(\ell_{i,3})$. Furthermore, if C_i is x_n-UNSAT, then $\mathcal{R}_{i|x_n} = -1$ and 0 otherwise. On the contrary, if C_i contains the negated literal $\bar{x}_n$, then $\mathcal{R}_{i|x_n} = g(\ell_{i,2})g(\ell_{i,3})$, with $\mathcal{R}_{i|x_n} = 1$ in the case that C_i is x_n-UNSAT.

We now connect these definitions with previous notions of *make* and *break* clauses. Note that a clause can only be a *make* or *break* with respect to x_n if it is x_n-UNSAT.

▶ **Observation 7.** *If $x_n = 1$, then all clauses with $\mathcal{R}_{i|x_n} = 1$ are makes, and all clauses with $\mathcal{R}_{i|x_n} = -1$ are breaks. If $x_n = 0$, then all clauses with $\mathcal{R}_{i|x_n} = -1$ are makes, and all clauses with $\mathcal{R}_{i|x_n} = 1$ are breaks.*

This leads us to the following Lemma:

▶ **Lemma 8.** *For some variable x_n,*

$$\sum_{x_n \in C_i} \mathcal{R}_{i|x_n} = (1 - 2x_n) \times (\mathcal{B}_n - \mathcal{M}_n) = \begin{cases} \mathcal{M}_n - \mathcal{B}_n & x_n = 1 \\ \mathcal{B}_n - \mathcal{M}_n & x_n = 0 \end{cases}$$

Before stating the final theorem, we need to introduce just the basic principle of how our proposed CDS system, cBRIM, works.

Cubic BRIM (cBRIM) design

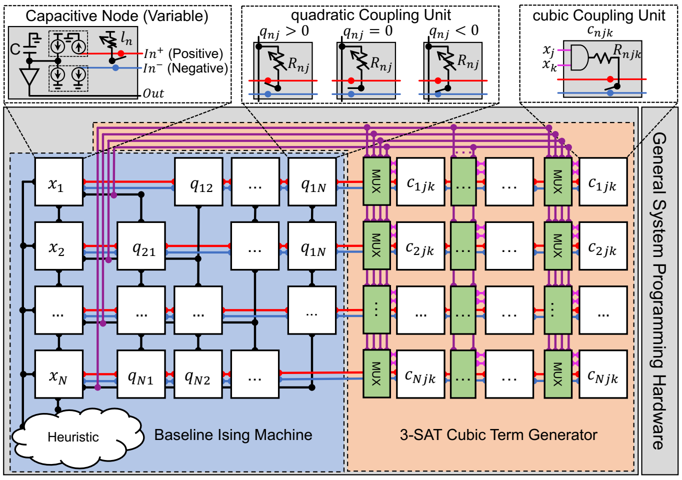


Figure 3 High level system overview of proposed cBRIM. The blue part is the baseline BRIM, while the orange part is new hardware to support cubic terms.

We extend BRIM [1, 46, 57] to support cubic terms for 3-SAT and refer to it as cubic BRIM or cBRIM in short. The full hardware details are beyond the scope of this paper and a number of variations in circuit design can achieve similar behavior at the system level. Here, we only discuss a high-level behavior model as shown in Fig. 3 and the pseudocode for simulation is shown in Appendix A. The general idea is to construct the hardware that naturally minimizes the cubic Hamiltonian H_C. The baseline system consists of an array of N bi-stable capacitive nodes whose voltages $\in [0, 1]$ represent the variables.[5] Nodes are interconnected by programmable resistive coupling units q. The resistance of a coupling unit connecting node n and node j, is given by:

[5] Note that the original BRIM uses a virtual ground $(\frac{V_{dd}+V_{ss}}{2})$. The voltage of the nodal capacitor ranges from V_{ss} to V_{dd}. Relative to the virtual ground, the polarity of the voltage serves as the spin representation (±1). In our current design, we do not use a virtual ground and the voltage quantizes to bit representation $\{0, 1\}$. This is because the bit representation is more convenient for SAT problems.

$$R_{nj} = \frac{R}{|q_{nj}|} \tag{7}$$

Where R is a constant resistance and q_{nj} is normalized to $[-1, +1]$. Thus, strong coupling means lower resistance. The sign of q_{nj} is implemented as follows: ① when $q_{nj} > 0$, the nodes are coupled in a parallel fashion (output of one node is connected to the positive input terminal of the other, via resistance R_{nj}: $Out_n \rightarrow In_j^+$, $Out_j \rightarrow In_n^+$); ② when $q_{nj} < 0$, they are coupled in an antiparallel fashion ($Out_n \rightarrow In_j^-$, $Out_j \rightarrow In_n^-$); ③ nodes are disconnected if $q_{nj} = 0$. Each node n can also have a linear bias l_n.

To support cubic terms, we add extra coupling units c. The idea is to pass voltages x_j and x_k through an AND gate to get $x_{jk} = x_j \times x_k$. Then we apply x_{jk} across a resistance of $R_{njk} = (\frac{R}{|c_{njk}|})$ and feed the resulting current to node n. The inputs to the AND gate are also programmable via multiplexers (MUX) depending on the 3-SAT formula. Now, applying Kirchhoff's current law, the rate of change of variable x_n is determined by the total incoming current via coupling resistors as described below:

$$\frac{dx_n}{dt} = \frac{1}{RC} \times (l_n + \sum_j q_{nj} x_j + \sum_{j<k} c_{njk} x_j x_k) \tag{8}$$

where C is the nodal capacitance. Moreover, as shown in Fig. 3, the variables can also be perturbed according to a selected heuristic of choice. If this perturbation is just a Gaussian white noise $\sim \mathcal{N}(0, \sigma^2)$, then the system is governed by Langevin dynamics (see Appendix B for more details). Now, consider the Hamiltonian H_C defined in Eq. 4. We can expand the summation and combine same degree polynomial terms together as shown below:

$$H_C = constant + \sum_n L_n x_n + \sum_{n<j} Q_{nj} x_n x_j + \sum_{n<j<k} C_{njk} x_n x_j x_k \tag{9}$$

where L_n, Q_{nj} and C_{njk} are respectively the coefficients of linear, quadratic, and cubic terms. If we take the gradient of H_C with respect to x_n, we get,

$$\frac{\partial H_C}{\partial x_n} = L_n + \sum_j Q_{nj} x_j + \sum_{j<k} C_{njk} x_j x_k \tag{10}$$

Comparing Eq. 8 and Eq. 10, if we set $l_n = -L_n$, $q_{nj} = -Q_{nj}$ and $c_{njk} = -C_{njk}$, then we can write,

$$\frac{dx_n}{dt} = -\alpha \frac{\partial H_C}{\partial x_n} \tag{11}$$

where $\alpha = \frac{1}{RC}$ is a known constant. Such a system is Lyapunov locally asymptotically stable [1] because it satisfies the criterion, $\nabla H_C \cdot \frac{dx}{dt} < 0$, unless x is a local minimum. Therefore, cBRIM naturally seeks a local minimum of H_C and stays there when found (unless perturbed by some heuristic). Moreover, using Definition 5 and the fact that x_n-Relevant forms are derived only from terms containing x_n, we can further write:

$$\frac{dx_n}{dt} = -\alpha \frac{\partial H_C}{\partial x_n} = -\alpha \sum_{x_n \in C_i} \mathcal{R}_{i|x_n} \tag{12}$$

Thus, the time dependent trajectory of each variable x_n is determined by the sum of all x_n-Relevant forms. Now, we can state the theorem that connects the system evolution of cBRIM with $\mathcal{M}_n/\mathcal{B}_n$ of each variable.

▶ **Theorem 9.** *Using Lemma 8 in Eq. 12,* $\frac{dx_n}{dt} = \alpha(1 - 2x_n) \times (\mathcal{M}_n - \mathcal{B}_n)$

From Eq. 8, cBRIM provides us the quantity $(\mathcal{M}_n - \mathcal{B}_n)$ in Theorem 9 as the total incoming current to each variable x_n. Thus, coupled with the hardware to generate $\mathcal{M}_n$ as in `cBRIM-BRW`, we can recover $\mathcal{B}_n$ information without explicitly considering other variable states. This simplifies the hardware and takes advantage of the dynamical interactions between nodes. We now propose a new heuristic to leverage this.

4.3.2 The tanh-make-break (TMB) heuristic

Motivated by the shortcomings of BRW as discussed in Section 4.2, we propose a heuristic that utilizes both $\mathcal{M}_n$ and $\mathcal{B}_n$. The probability to flip a variable x_n is given by Eq. 13 using two nonlinear functions f and h:

$$p(x_n) = f(x_n) \cdot h(x_n) \tag{13}$$

In our testing, we selected $f(x_n) = \tanh(c_m \cdot \mathcal{M}_n)$ and $h(x_n) = 1 - \tanh(c_b \cdot \mathcal{B}_n)$. Here, $c_m > 0$ and $c_b > 0$ are the *make* and *break* coefficients respectively. A number of nonlinear functions could work, and we leave exhaustive exploration as future work. The motivation behind the choice of tanh is threefold:

- Convenience: Given the range ($\tanh(y) \in [0, 1]$ for $y \geq 0$), the product $f(x_n) \cdot h(x_n)$ is mathematically a convenient probability without any need for normalization. Physically, it is easy to design a simple circuit with tanh-like behavior.
- SAT preservation: If all clauses are SAT, then $\tanh(c_m \cdot \mathcal{M}_n) = 0$ for all x_n. Therefore, the system will not be perturbed once a satisfying state is reached.
- Nonlinearity: Previous heuristics using $\mathcal{M}_n$ and $\mathcal{B}_n$ utilized nonlinear functions with a steep slope [8, 48]. It was shown to significantly outperform linear functions. Therefore, we seek to preserve this characteristic.

We combine this *tanh-make-break* (TMB) heuristic with cBRIM and refer to this solver as `cBRIM-TMB`. Empirical observations suggest TMB heuristic is not overly sensitive to parameters. For instance, $c_m = 0.9$, $c_b = 0.4$ works best for the tested scale-free problems while for uniform random problems, $c_m = 0.9$, $c_b = 0.6$ works the best. Moreover, we found that the same c_m and c_b parameters worked for both 500- and 1000-variable problems.

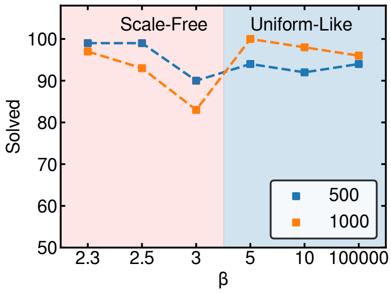


Figure 4 The performance of `cBRIM-TMB` tested on the same problems as in Fig. 2 and the same cutoff times. The plot shows the number of problems solved (out of 100) for varying values of β.

Fig. 4 demonstrates the performance of `cBRIM-TMB` which can be directly compared against that of `cBRIM-BRW` shown in Fig. 2. Unlike `cBRIM-BRW`, `cBRIM-TMB` is able to solve most of the industrial-like scale-free problems as well as uniform random problems with only a minor dip in performance when $\beta \approx 3$. Moreover, it is able to maintain high success rates using the same parameters for both 500 and 1000 variable problems, indicating a weaker dependence on problem size for parameter tuning unlike `cBRIM-BRW`.

We leave it as future work to analyze the time dependent solver behavior, and to relate the performance of heuristics with problem structure. In the next section, we compare `cBRIM-TMB` with state-of-the-art CDCL and SLS solvers on generated scale-free and uniform random problems to estimate its real-world efficacy.

5 Evaluation

We now compare the performance of `cBRIM-BRW` and our proposed `cBRIM-TMB` against three state-of-the-art solvers: one CDCL based: KISSAT [14] and two SLS based: WalkSAT [34] and ProbSAT [8]. CDCL solvers have consistently outperformed other paradigms in SAT competitions, particularly in large industrial benchmarks. KISSAT and its variants are paradigmatic of high performance CDCL, placing top in recent SAT competitions [10, 14]. WalkSAT is representative of flip-efficient SLS heuristics, both in standalone solvers and those integrated into hybrid CDCL-SLS approaches [34]. Comparing against both algorithms will provide further context in which to evaluate `cBRIM-TMB`'s performance. We also choose to compare with the original ProbSAT solver since our heuristic is derived from it.

5.1 Methodology

Results of all software solvers: KISSAT [14], WalkSAT [34], and ProbSAT [8] are collected on an Intel Xeon Platinum 8268 CPU running at 2.90GHz. Each process is granted 8 GB of memory. KISSAT is allowed to run with a timeout of 10,000 seconds. WalkSAT is run with a 5 million flips cutoff with 20 retries using the `-best` heuristic (otherwise known as "SKC") [45]. ProbSAT is used with its default configuration, with no cutoff specified. `cBRIM-TMB` and `cBRIM-BRW` are simulated by solving differential equations that describe the system dynamics. Each problem is simulated 20 times with different initial conditions. We start with 1 ms simulation time, up to 200 ms for unsolved instances.

A 200-problem suite of 1000-variable instances were generated using a tool developed by Ansótegui et al. [4]. 100 are uniform random with $M/N = 4.25$ (4250 clauses) and 100 are scale-free with $M/N = 3.4$ (3400 clauses) and $\beta = 2.935$, which is observed to produce satisfiable problems about 50% of the time, with many being "hard" instances. All problems are verified as satisfiable using either WalkSAT or KISSAT. Instances are limited to 1000 variables and `cBRIM` run times are kept short due to the large simulation overhead of CDS (about $10^5\times$ slower than real hardware time).

For the reported solver time and flips, we use a "metric-to-solution" (MTS) formula, which is a generalization of "time-to-solution" [29] and is given by:

$$MTS = \begin{cases} m \times \dfrac{\log_{10}(0.01)}{\log_{10}(1-S)} & S < 0.99 \\ m & \text{otherwise} \end{cases} \tag{14}$$

where, m is the average value of the metric of interest (time or flips) and S is the success probability of finding a satisfying solution for a single problem. Thus, MTS tells us the estimated metric value needed to find a solution with 99% probability, consequently penalizing solvers finding solutions with low success rate. The reported flips and times for WalkSAT, `cBRIM-TMB`, and `cBRIM-BRW` are then "flips-to-solution"(FTS) and "time-to-solution" (TTS) respectively. The flip counts used to calculate FTS for `cBRIM-TMB` and `cBRIM-BRW` are the sum of "heuristic flips" and "natural flips". The latter are purely due to system dynamics, excluding that of the added heuristic.

5.2 Results

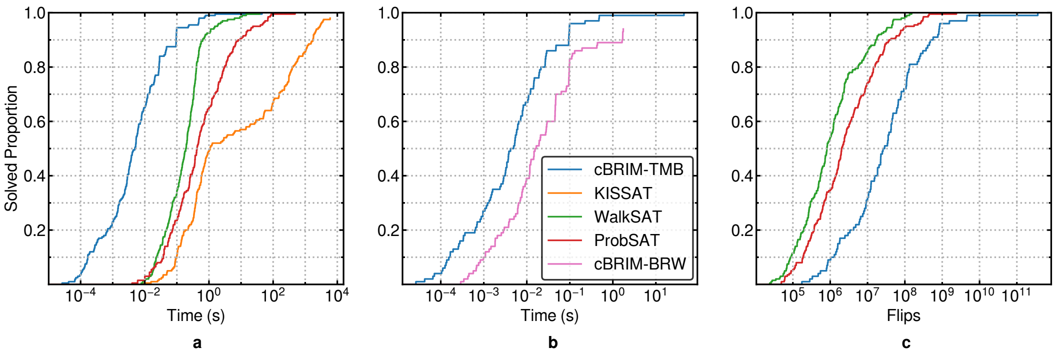


Figure 5 Solver performance comparisons on 200 randomly generated 1000-variable problems: 100 scale-free with $M = 3400$ and 100 uniform random with $M = 4250$. (a) The proportion of problem instances solved by each solver as time increases. Reported times for cBRIM-TMB and WalkSAT are calculated using the MTS formula in Eq. 14. (b) Comparing the performance of cBRIM-TMB and cBRIM-BRW on the 100 scale-free problems as time increases. (c) The same plot but with increasing flip count for the SLS solvers and cBRIM-TMB. Flip counts are similarly scaled according to Eq. 14.

First, we tested cBRIM with a problem-agnostic annealing schedule which could only solve 7% of the 200 problems with a TTS of 0.59 s in the worst case. Due to this poor performance and the requirement of ever longer run times, we have decided to remove it from the main analysis. Fig. 5a shows the proportion of problems solved by cBRIM-TMB and the software solvers versus execution time. cBRIM-TMB and the SLS solvers were able to solve all 200 problems. In contrast, KISSAT is able to solve all scale-free problems quickly, but is unable to solve 4 uniform random problems before the cutoff. Both SLS solvers outperformed KISSAT in the majority of the instances. However, 23 of the 100 scale-free problems are solved faster by KISSAT, with a geometric mean speedup of 2×. This indicates that some aspects of industrial problem structure is captured in the benchmark suite which KISSAT is able to exploit.

We also observe that for all the solvers, the curves flatten as we increase the run time. For cBRIM-TMB, 199 of the 200 problems could be solved in just 1.56 s in the worst-case, while the remaining scale-free problem took 28.8× longer. WalkSAT and ProbSAT have a similar trend in terms of run time increase, but it is not as dramatic. For instance, 199 problems took 14.53 seconds and 86.84 seconds respectively, while the last problem took 2.6× and 5.6× longer respectively. Like cBRIM-TMB, ProbSAT struggled with a scale-free problem while a uniform random problem proved difficult for WalkSAT. KISSAT could solve 50% of the instances within 1 s with majority of them being scale-free. The uniform random problems that it could solve, took orders of magnitude longer in the worst-case scenario. It is worth noting that larger scale-free problems likely favor CDCL solvers over SLS counterparts [10]. Therefore, the performance picture of CDS such as cBRIM-TMB for large real industrial instances is incomplete. Nevertheless, the results in Fig. 5a portray CDS as a promising avenue for execution time reduction.

Next, we compare the two versions of cBRIM. We find that both cBRIM-TMB and cBRIM-BRW perform similarly well for uniform random problems (except that cBRIM-BRW could not solve 1 problem instance). Therefore, Fig. 5b only shows the performance on the 100 scale-free problems with increasing run time. We can observe that cBRIM-TMB is consistently better than cBRIM-BRW by about 4× and moreover, the latter could not solve 6 out of the 100 scale-free problems. This emphasizes the importance of considering $\mathcal{B}_n$ for industrial-like problems.

Another interesting analysis is to look at how efficiently these solvers arrive at a solution. Fig. 5c shows the proportion of problems solved as flip counts increase for `cBRIM-TMB` and the SLS solvers. Both WalkSAT and ProbSAT do outperform `cBRIM-TMB`. WalkSAT in particular stands out as the most "flip-efficient". This is to be expected: a system doing parallel flips can effect many more state changes than sequential algorithms. Several variables can flip in a single move, compared to just one for WalkSAT or ProbSAT. The redundancy of parallel flip schemes with respect to $\mathcal{M}_n/\mathcal{B}_n$ is another area for future study and optimization. In any case, the extremely fast dynamics of `cBRIM-TMB` more than compensates for higher flip counts which results in its observed speedup. This will become even clearer in the following analysis.

Table 1 Summary of the results shown in Figure 5 for each problem type. The time and flip values are the geometric mean of all successful solutions for each solver scaled by the MTS formula in Eq. 14.

Solver	Dist	Solved	Time (s)	Flips	Flip-rate (Flips/μs)
`cBRIM-TMB`	Scale-Free	100/100	4.06e-03	2.78e+07	6853.00
	Uniform	100/100	5.41e-03	9.34e+06	1726.00
`cBRIM-BRW`	Scale-Free	94/100	1.58e-02	4.00e+06	252.58
	Uniform	99/100	6.86e-03	8.62e+06	1255.88
KISSAT	Scale-Free	100/100	2.50e-01	N/A	N/A
	Uniform	96/100	1.62e+02	N/A	N/A
WalkSAT	Scale-Free	100/100	1.03e-01	5.23e+05	5.23
	Uniform	100/100	2.59e-01	1.96e+06	7.54
ProbSAT	Scale-Free	100/100	6.01e-01	3.2e+06	5.32
	Uniform	100/100	4.30e-01	2.05e+06	4.77

Table 1 summarizes the results of each solver's performance separately on scale-free and uniform random problems. The reported times and flips are the geometric mean of the respective metric for all successful runs scaled by the MTS formula in Eq. 14. As an example, let us take the case of scale-free problems. `cBRIM-TMB` requires about 53× and 8.7× more flips than WalkSAT and ProbSAT respectively. However, it flips over 550× faster than both the SLS solvers in the geometric average case (see "Flip-rate" in table) which results in `cBRIM-TMB` having a net speedup of 25× and 147× over WalkSAT and ProbSAT respectively. Fast hardware dynamics enable a high "Flip-rate" for `cBRIM-TMB` which takes about 0.5 ns to flip a variable, compared to numerous instructions per flip for software solvers [46].

KISSAT outperforms ProbSAT by 2.4× in run time for scale-free problems, but falls behind both `cBRIM-TMB` and WalkSAT. As expected, uniform random problems proved particularly difficult for KISSAT. Such problems lack structure that a CDCL solver like KISSAT can exploit [18, 37].

6 Conclusions and future work

Non-von Neumann computing using dynamical systems show potential for extreme speedup and energy efficiencies for difficult optimization problems. Most of these solvers operate on quadratic models like Ising/QUBO. Recently, cubic dynamical solvers (CDS) have also been proposed for problems like SAT. In this paper, we show that such CDS systems that rely on problem-agnostic heuristics do not seem to offer any tangible benefit over software SAT solvers. This motivates us to utilize SAT-specific heuristics. The combination of a *make*-only heuristic with CDS does perform well for uniform random problems but underperforms for industrial-like, scale-free instances. Our analysis lead us to the opinion that $\mathcal{M}_n$ alone might be insufficient for such skewed variable distribution in problems and we need to take into account $\mathcal{B}_n$ as well. In the process, we derive a novel relationship between the time derivative of each variable in CDS to its $(\mathcal{M}_n - \mathcal{B}_n)$ quantity. Given that $\mathcal{M}_n$ can be computed easily, one can recover $\mathcal{B}_n$ using this relationship. This allows us to propose a new heuristic using a nonlinear tanh-based function of $\mathcal{M}_n$ and $\mathcal{B}_n$ which we combine with CDS. With simulation results, our proposed solver, called `cBRIM-TMB`, is projected to perform orders of magnitude faster than software SAT solvers both in scale-free and uniform random problems. The speedup can be attributed to the extremely fast dynamics of CDS-based systems that give rise to a high flip-rate.

We believe that `cBRIM-TMB` is just a first attempt at combining a *make/break* heuristic with a CDS system. Further design space exploration is likely to come up with more efficient solution systems. Testing on real SAT instances is also needed to judge the true effectiveness of our solver especially because a reduction from generic k-SAT to 3-SAT is required. Moreover, since real SAT instances are huge (containing hundreds of thousands of variables), efficient software-hardware co-design approach is necessary owing to the limited capacity of hardware. We hope that our proposed solver can make CDS a promising paradigm to solve SAT and encourage more people to contribute to it.

References

1 Richard Afoakwa, Yiqiao Zhang, Uday Kumar Reddy Vengalam, Zeljko Ignjatovic, and Michael Huang. Brim: Bistable resistively-coupled Ising machine. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 749–760, 2021. doi:10.1109/HPCA51647.2021.00068.

2 Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Rev. Mod. Phys.*, 90:015002, January 2018. doi:10.1103/RevModPhys.90.015002.

3 Tasniem Nasser Alyahya, Mohamed El Bachir Menai, and Hassan Mathkour. On the structure of the boolean satisfiability problem: A survey. *ACM Computing Surveys (CSUR)*, 55(3):1–34, 2022.

4 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. On the structure of industrial SAT instances. In *Principles and Practice of Constraint Programming-CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20-24, 2009 Proceedings 15*, pages 127–141. Springer, 2009.

5 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Towards industrial-like random SAT instances. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, pages 387–392, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

6 Masashi Aono, Makoto Naruse, Song-Ju Kim, Masamitsu Wakabayashi, Hirokazu Hori, Motoichi Ohtsu, and Masahiko Hara. Amoeba-inspired nanoarchitectonic computing: Solving intractable computational problems using nanoscale photoexcitation transfer dynamics. *Langmuir : the ACS journal of surfaces and colloids*, 29, April 2013. doi:10.1021/la400301p.

7 Daniel Azses, Maxime Dupont, Bram Evert, Matthew J. Reagor, and Emanuele G. Dalla Torre. Navigating the noise-depth tradeoff in adiabatic quantum circuits, 2022. doi:10.48550/arXiv.2209.11245.

8 Adrian Balint and Uwe Schöning. Choosing probability distributions for stochastic local search and the role of make versus break. In *SAT*, 2012.

9 Mohammad Khairul Bashar, Antik Mallick, Avik W. Ghosh, and Nikhil Shukla. Dynamical system-based computational models for solving combinatorial optimization on hypergraphs. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, pages 1–1, 2023. doi:10.1109/JXCDC.2023.3235113.

10 Tomas Baylo, Marijn J.H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Proceedings of SAT competition 2022: Solver and benchmark descriptions. In *Proceedings of SAT Competition 2022 : Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, Helsinki, 2022. Department of Computer Science, University of Helsinki. URL: `http://hdl.handle.net/10138/347211`.

11 Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. Noisy intermediate-scale quantum algorithms. *Rev. Mod. Phys.*, 94:015004, February 2022. `doi:10.1103/RevModPhys.94.015004`.

12 Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving sat (and maxsat) with a quantum annealer: Foundations, encodings, and preliminary results. *Information and Computation*, 275:104609, 2020. `doi:10.1016/j.ic.2020.104609`.

13 A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications.* IOS Press, NLD, 2009.

14 Armin Biere and Mathias Fleury. Gimsatul, IsaSAT, Kissat entering the SAT competition 2022. In *Proceedings of SAT Competition 2022 : Solver and Benchmark Descriptions*, 2022.

15 Sami Boulebnane and Ashley Montanaro. Solving boolean satisfiability problems with the quantum approximate optimization algorithm, 2022. `doi:10.48550/arXiv.2208.06909`.

16 Connor Bybee, Denis Kleyko, Dmitri E Nikonov, Amir Khosrowshahi, Bruno A Olshausen, and Friedrich T Sommer. Efficient optimization with higher-order Ising machines. *arXiv preprint*, 2022. `arXiv:2212.03426`.

17 Muya Chang, Xunzhao Yin, Zoltan Toroczkai, Xiaobo Hu, and Arijit Raychowdhury. An analog clock-free compute fabric base on continuous-time dynamical system for solving combinatorial optimization problems. In *2022 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–2, 2022. `doi:10.1109/CICC53496.2022.9772850`.

18 Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. Combining vsids and chb using restarts in SAT. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

19 Tzuu-Shuh Chiang, Chii-Ruey Hwang, and Shuenn Jyi Sheu. Diffusion for global optimization in $\mathbb{R}^n$. *SIAM Journal on Control and Optimization*, 25(3):737–753, 1987. `doi:10.1137/0325042`.

20 D-WAVE. minorminer, 2014. URL: `https://github.com/dwavesystems/minorminer`.

21 D-WAVE. Operation and timing, 2022. URL: `https://docs.dwavesys.com/docs/latest/c_qpu_timing.html`.

22 John D. Davis, Zhangxi Tan, Fang Yu, and Lintao Zhang. A practical reconfigurable hardware accelerator for boolean satisfiability solvers. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 780–785, New York, NY, USA, 2008. Association for Computing Machinery. `doi:10.1145/1391469.1391669`.

23 Mária Ercsey-Ravasz and Zoltán Toroczkai. Optimization hardness as transient chaos in an analog approach to constraint satisfaction. *Nature Physics*, 7(12):966–970, 2011.

24 Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014. `doi:10.48550/arXiv.1411.4028`.

25 D. Freedman and P. Drineas. Energy minimization via graph cuts: settling what is possible. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 939–946 vol. 2, 2005. `doi:10.1109/CVPR.2005.143`.

26 Thomas Gabor, Sebastian Zielinski, Sebastian Feld, Christoph Roch, Christian Seidel, Florian Neukart, Isabella Galter, Wolfgang Mauerer, and Claudia Linnhoff-Popien. Assessing solution quality of 3SAT on a quantum annealing platform, 2019. `doi:10.48550/arXiv.1902.04703`.

27 Hayato Goto, Kosuke Tatsumura, and Alexander R Dixon. Combinatorial optimization by simulating adiabatic bifurcations in nonlinear hamiltonian systems. *Science advances*, 5(4):eaav2372, 2019.

28 Kanupriya Gulati, Suganth Paul, Sunil P. Khatri, Srinivas Patil, and Abhijit Jas. Fpga-based hardware acceleration for boolean satisfiability. *ACM Trans. Des. Autom. Electron. Syst.*, 14(2), April 2009. `doi:10.1145/1497561.1497576`.

29 Ryan Hamerly, Takahiro Inagaki, Peter L. McMahon, Davide Venturelli, Alireza Marandi, Tatsuhiro Onodera, Edwin Ng, Carsten Langrock, Kensuke Inaba, Toshimori Honjo, Koji Enbutsu, Takeshi Umeki, Ryoichi Kasahara, Shoko Utsunomiya, Satoshi Kako, Ken ichi Kawarabayashi, Robert L. Byer, Martin M. Fejer, Hideo Mabuchi, Dirk Englund, Eleanor Rieffel, Hiroki Takesue, and Yoshihisa Yamamoto. Experimental investigation of performance differences between coherent Ising machines and a quantum annealer. *Science Advances*, 5(5):eaau0823, 2019. `doi:10.1126/sciadv.aau0823`.

30 Kazuaki Hara, Naoki Takeuchi, Masashi Aono, and Yuko Hara-Azumi. Amoeba-inspired stochastic hardware sat solver. In *20th International Symposium on Quality Electronic Design (ISQED)*, pages 151–156, 2019. `doi:10.1109/ISQED.2019.8697729`.

31 Takahiro Inagaki, Yoshitaka Haribara, Koji Igarashi, Tomohiro Sonobe, Shuhei Tamate, Toshimori Honjo, Alireza Marandi, Peter L McMahon, Takeshi Umeki, Koji Enbutsu, et al. A coherent Ising machine for 2000-node optimization problems. *Science*, 354(6312):603–606, 2016.

32 Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse Ising model. *Physical Review E*, 58(5):5355–5363, November 1998. `doi:10.1103/physreve.58.5355`.

33 Taro Kanao and Hayato Goto. Simulated bifurcation for higher-order cost functions. *Applied Physics Express*, 16(1):014501, 2022.

34 Henry Kautz. Walksat, 2018. URL: `https://gitlab.com/HenryKautz/Walksat`.

35 Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

36 V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004. `doi:10.1109/TPAMI.2004.1262177`.

37 Jia Hui Liang, Vijay Ganesh, Ed Zulkoski, Atulan Zaman, and Krzysztof Czarnecki. Understanding vsids branching heuristics in conflict-driven clause-learning SAT solvers. In *Hardware and Software: Verification and Testing: 11th International Haifa Verification Conference, HVC 2015, Haifa, Israel, November 17-19, 2015, Proceedings 11*, pages 225–241. Springer, 2015.

38 Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2:5, 2014.

39 Debasis Mitra, Fabio Romeo, and Alberto Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. In *1985 24th IEEE Conference on Decision and Control*, pages 761–767, 1985. `doi:10.1109/CDC.1985.268600`.

40 Ferenc Molnár, Shubha R Kharel, Xiaobo Sharon Hu, and Zoltán Toroczkai. Accelerating a continuous-time analog SAT solver using gpus. *Computer Physics Communications*, 256:107469, 2020.

41 Anh Hoang Ngoc Nguyen, Masashi Aono, and Yuko Hara-Azumi. Amoeba-inspired hardware sat solver with effective feedback control. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 243–246, 2019. `doi:10.1109/ICFPT47387.2019.00038`.

42 Tiemo Pedergnana and Nicolas Noiray. Exact potentials in multivariate langevin equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(12):123146, 2022. `doi:10.1063/5.0124031`.

43 Irina Rish and Rina Dechter. Resolution versus search: Two strategies for SAT. *Journal of Automated Reasoning*, 24(1-2):225–275, 2000.

44 I. G. Rosenberg. Reduction of bivalent maximization to the quadratic case. *Cahiers du Centre d'Etudes de Recherche Operationnelle*, 17:71–74, 1975.

45 Bart Selman, Henry A Kautz, Bram Cohen, et al. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability*, 26:521–532, 1993.

46 Anshujit Sharma, Richard Afoakwa, Zeljko Ignjatovic, and Michael Huang. Increasing Ising machine capacity with multi-chip architectures. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, pages 508–521, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3470496.3527414.

47 Anshujit Sharma, Matthew Burns, Andrew Hahn, and Michael Huang. Augmented electronic ising machine as an effective sat solver, 2023. arXiv:2305.01623.

48 Yong Shim, Abhronil Sengupta, and Kaushik Roy. Biased random walk using stochastic switching of nanomagnets: Application to SAT solver. *IEEE Transactions on Electron Devices*, 65(4):1617–1624, 2018. doi:10.1109/TED.2018.2808232.

49 Ali Asgar Sohanghpurwala and Peter Athanas. An effective probability distribution SAT solver on reconfigurable hardware. In *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, 2016.

50 Ali Asgar Sohanghpurwala, Mohamed W. Hassan, and Peter M. Athanas. Hardware accelerated SAT solvers - a survey. *J. Parallel Distributed Comput.*, 106:170–184, 2017.

51 Juexiao Su, Tianheng Tu, and Lei He. A quantum annealing approach for boolean satisfiability problem. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

52 Tianshi Wang, Leon Wu, and Jaijeet Roychowdhury. New computational results and hardware prototypes for oscillator-based Ising machines. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3316781.3322473.

53 Yulun Wang and Predrag S. Krstic. Prospect of using grover's search in the noisy-intermediate-scale quantum-computer era. *Physical Review A*, 102(4), October 2020. doi:10.1103/physreva.102.042609.

54 Ryan Williams. *Maximum Two-Satisfiability*, pages 507–510. Springer US, Boston, MA, 2008. doi:10.1007/978-0-387-30162-4_227.

55 Pan Xu, Jinghui Chen, Difan Zou, and Quanquan Gu. Global convergence of langevin dynamics based algorithms for nonconvex optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

56 Xunzhao Yin, Behnam Sedighi, Melinda Varga, Maria Ercsey-Ravasz, Zoltan Toroczkai, and Xiaobo Sharon Hu. Efficient analog circuits for boolean satisfiability. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(1):155–167, 2017.

57 Yiqiao Zhang, Uday Kumar Reddy Vengalam, Anshujit Sharma, Michael Huang, and Zeljko Ignjatovic. Qubrim: A cmos compatible resistively-coupled Ising machine with quantized nodal interactions. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3508352.3549443.

A Cubic BRIM (cBRIM) Simulation

For simulation purposes, we consider the product voltage x_{jk} as an auxiliary variable and incorporate all the coefficients l_n, q_{nj} and c_{njk} into one matrix J. We simulate cBRIM using the following pseudocode:

```
// f: the 3SAT formula
// simtime: the total anneal time,
// tstep: the simulation step size
procedure cBRIM (f, simtime, tstep) {
    // Initialize the coupling matrix and variable
    // based on the problem
    J_matrix, x_vector = initialize_problem(f);

    // Update the auxiliary variable, x_jk = x_j * x_k
    update_auxiliary(x_vector);
    // Initialize time to 0
    time = 0;

    // Main simulation loop
    while(time < simtime) {
        // Get the quantized variables in {0,1}
        qx_vector = Quant(x_vector);

        // Compute the derivatives of variables
        // by matrix vector multiplication
        dxdt = (J_matrix * qx_vector)/(R * C);

        // Select heuristic spin flips
        hflips = heuristic(f, time, qx_vector);

        // Assign opposing currents to perform heuristic flips
        // hfR is heuristic flip resistance (1KOhm)
        for (var_id in hflips) {
            dxdt(var_id) =
                (!hflips(var_id) - x_vector(var_id)) / (hfR * C);
        }

        // Update the voltages of variables
        x_vector += dxdt * tstep;

        // Limit voltages in [0, 1]
        limit(x_vector);

        // Update the auxiliary variables
        update_auxiliary(x_vector);

        // Increment time
        time += tstep;
    }

    // return quantized state, 1 for true and 0 for false
    return Quant(x_vector);
}
```

B Langevin cBRIM

Here, we give a theoretical analysis of using a simple Gaussian white noise perturbation $\eta \sim \mathcal{N}(0, \sigma^2)$ in cBRIM. We can rewrite Eq. 8 as follows:

$$\frac{dx_n}{dt} = \frac{1}{RC} \times (l_n + \sum_j q_{nj} x_j + \sum_{j<k} c_{njk} x_j x_k) + \eta \tag{15}$$

Following a similar procedure as in Eq. 9 to Eq. 11, we arrive at the following expression:

$$\frac{dx_n}{dt} = -\alpha \frac{\partial H_C}{\partial x_n} + \eta \tag{16}$$

where $\alpha = \frac{1}{RC}$. Such a system is governed by Langevin dynamics and the joint probability density $P(x, t)$ for some state x evolves according to the Fokker-Planck equation [42]:

$$\frac{\partial P(x,t)}{\partial t} = \nabla \cdot \left[\alpha P(x,t) \nabla H_C + \frac{\sigma^2}{2} \nabla P(x,t) \right] \tag{17}$$

We can derive the stationary distribution when $t \to \infty$ by setting $\frac{\partial P(x,t)}{\partial t} = 0$ in Eq. 17. This results in the boltzmann distribution as intended [19, 55]:

$$P_\infty(x) = \frac{1}{Z} \exp \left[\frac{-2\alpha H_C(x)}{\sigma^2} \right] \tag{18}$$

where Z is the normalization constant. Eq. 18 implies that when the system settles into equilibrium, the ground state(s) have higher probability than other states.

Cutting Planes Width and the Complexity of Graph Isomorphism Refutations

Jacobo Torán
Institut für Theoretische Informatik, Universität Ulm, Germany

Florian Wörz
Institut für Theoretische Informatik, Universität Ulm, Germany

Abstract

The width complexity measure plays a central role in Resolution and other propositional proof systems like Polynomial Calculus (under the name of degree). The study of width lower bounds is the most extended method for proving size lower bounds, and it is known that for these systems, proofs with small width also imply the existence of proofs with small size. Not much has been studied, however, about the width parameter in the Cutting Planes (CP) proof system, a measure that was introduced by Dantchev and Martin in 2011 under the name of CP cutwidth.

In this paper, we study the width complexity of CP refutations of graph isomorphism formulas. For a pair of non-isomorphic graphs G and H, we show a direct connection between the Weisfeiler–Leman differentiation number $\mathsf{WL}(G,H)$ of the graphs and the width of a CP refutation for the corresponding isomorphism formula $\mathrm{Iso}(G,H)$. In particular, we show that if $\mathsf{WL}(G,H) \leq k$, then there is a CP refutation of $\mathrm{Iso}(G,H)$ with width k, and if $\mathsf{WL}(G,H) > k$, then there are no CP refutations of $\mathrm{Iso}(G,H)$ with width $k-2$. Similar results are known for other proof systems, like Resolution, Sherali–Adams, or Polynomial Calculus. We also obtain polynomial-size CP refutations from our width bound for isomorphism formulas for graphs with constant WL-dimension.

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Complexity theory and logic; Mathematics of computing → Graph theory

Keywords and phrases Cutting Planes, Proof Complexity, Linear Programming, Combinatorial Optimization, Weisfeiler–Leman Algorithm, Graph Isomorphism

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.26

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2023/063/`

Funding Supported by the Deutsche Forschungsgemeinschaft (DFG) under project number 430150230, “Complexity measures for solving propositional formulas”.

1 Introduction

Central to the field of combinatorial optimization is the NP-hard problem of finding *integer* solutions to linear programs. This is done by optimizing the linear objective function $\langle \mathbf{c}, \mathbf{x} \rangle$ (for a given vector $\mathbf{c} \in \mathbb{R}^n$) over the set of feasible points $\mathbf{x}$ for the *LP relaxation*, described by a rational polytope of the form

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \geq \mathbf{b},\ \mathbf{0} \leq \mathbf{x} \leq \mathbf{1}\},$$

where $\mathbf{A} \in \mathbb{Z}^{m \times n}$ is some integer matrix, and $\mathbf{b} \in \mathbb{Z}^m$ is an integer vector.[1] If the polytope is *integral* (i. e., only contains integer vertices), one can optimize over all *real* vectors in P (i. e., solve the linear relaxation in weakly polynomial time). Otherwise, one has to consider the *integral hull* $P^{\mathbb{Z}} := \mathrm{conv}(P \cap \mathbb{Z}^n)$ for the optimization, i. e., the smallest polytope containing

[1] Note that we restrict our attention to polytopes in $[0,1]^n$ rather than polyhedrons in $\mathbb{R}^n$.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 26; pp. 26:1–26:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the integral points of P. As was already suggested by Gomory [22] and later by Chvátal [12], in such a case, one can iteratively refine the set of feasible solutions by adding further valid constraints described by hyperplanes, or, more precisely, half-spaces, to the set of inequalities describing P. These half spaces still contain $P^{\mathbb{Z}}$ but – hopefully – cut off some parts of P. For this purpose, the *cut rule* adds an inequality of the form $\langle \mathbf{a}, \mathbf{x} \rangle \geq \lceil b \rceil$ with an integral vector $\mathbf{a}$ and a rational number b such that every point of P satisfies the inequality $\langle \mathbf{a}, \mathbf{x} \rangle \geq b$. If b is not an integer, then the former inequality is not valid for (some) fractional solutions but still valid for all integer solutions. This process yields a sequence

$$P \supseteq P^{(1)} \supseteq P^{(2)} \supseteq \cdots \supseteq P^{\mathbb{Z}}$$

of polytopes. If some polytope $P^{(i)}$ in this sequence is empty, P cannot have integer solutions.

Cutting Planes Proof System. Using this idea, Cook et al. [16] introduced the Cutting Planes proof system. In this system, one is initially given a set $\left\{ \sum_{j=1}^{n} a_{i,j} x_j \geq b_i \;\middle|\; i \in [m] \right\}$ of integer inequalities describing the polytope P. Using the two deduction rules introduced in Definition 2, one can repeatedly deduce new inequalities, aiming to derive the contradictory inequality $0 \geq 1$. Obtaining a sequence of inequalities ending with $0 \geq 1$ is possible if and only if the initial set of inequalities does not admit an integer solution. This yields the Cutting Planes proof system (formally introduced in Section 2.2).

In particular, CP can be used to refute unsatisfiable CNF formulas (by translating them into affine inequalities). Cutting Planes is a strong proof system that can simulate Resolution, and it is exponentially stronger for several formula classes [16]. Exponential lower bounds on the size of a Cutting Planes proof (as measured in the number of inequalities) have been shown using the interpolation method [45, 27, 30] and, more recently, using lifting and communication complexity results [21] that can be traced back to [34, 9, 31].

Other complexity measures for CP have been studied. These measures are defined by the directed acyclic graph representing the proof (one connects the premises with the consequences). The *rank* of a proof is the maximum number of applications of the cut rule along any path in the directed graph. This is known as the *Chvátal rank* in linear optimization (see [36] for an excellent overview in this area) and was introduced in [10] in the area of proof complexity. This measure is the analogon of *depth* in Resolution [54]. Further, Dantchev and Martin [18] introduced the parameter *cutwidth*, defined as the maximum number of variables present in an inequality derived by performing a cut. This measure was further studied in [46] under the name of width, where the author presents linear lower bounds for this measure, as well as width/rank tradeoffs. In the case of Resolution, there is also a related complexity measure of *width* that measures how many literals are present in the largest clause in a refutation. In Polynomial Calculus, the analogous measure is *degree*. The seminal paper [5] showed that proving width lower bounds for Resolution is a way to prove size lower bounds for Resolution. This result extends to the corresponding measures in Polynomial Calculus [13, 35]. These papers sparked interest in the width/degree complexity measures, resulting in a long line of papers proving lower bounds for these measures. The situation for CP width lower bounds is dramatically more sparse. We are only aware of the two mentioned references [18, 46].

In this paper, we study graph isomorphism formulas with respect to the parameters rank and width. This allows us to prove size upper bounds for isomorphism formulas based on graphs with constant Weisfeiler–Leman dimension. We also show lower bounds for these formulas in a subsystem of CP. A strong motivation for this study is that Cutting Planes is a promising candidate to be used in future efficient implementations of SAT solvers.

Furthermore, proof complexity results also hold for all integer linear programming solvers based on the Gomory–Chvátal rule. These solvers provide up-to-date methods for solving NP-hard Boolean optimization problems.

Weisfeiler–Leman and Proof Complexity. The graph isomorphism problem (GI), i. e., the task of deciding whether two given graphs are isomorphic, has been intensively studied and is well known for its unresolved complexity, as it is one of the few problems in NP that is not known to be complete for this class nor to be in P. It is also unknown whether GI $\in$ co-NP.

A naïve heuristic to distinguish two non-isomorphic colored graphs is the 1-dimensional Weisfeiler–Leman algorithm (WL), or color refinement algorithm. This algorithm updates the original vertex colors according to the multiset of colors of their neighbors. This basic step is applied repeatedly until the colorings stabilize. This procedure can be generalized to the k-dimensional Weisfeiler–Leman algorithm (k-WL) [56, 55]. In this more refined variant, the set of k-tuples of vertices is partitioned into automorphism-invariant equivalence classes (see, e. g., [37] for an overview of this procedure). It had been conjectured that GI is solvable using the k-dimensional Weisfeiler–Leman algorithm, with k being sublinear in the number of vertices of the graphs. However, this was shown to be false in the seminal work of Cai, Fürer, and Immerman [11]. Fascinatingly, the authors achieved this by relating the power of k-WL to the expressive power of $\mathcal{C}^k$, the k-variable fragment of first-order logic augmented with counting quantifiers, and a variant of an Ehrenfeucht-Fraïssé game [20, 19] called the bijective k-pebble game. Nevertheless, the Weisfeiler–Leman method still plays a central role in the algorithmic research on GI; for example, Babai's famous algorithm for GI [3] uses the Weisfeiler–Leman method as a subroutine.

The field of proof complexity provides a different approach to studying the complexity of the GI problem. Here, one tries to find the smallest size of a proof of the fact that two graphs are non-isomorphic. It holds that GI is in co-NP if and only if there is a concrete proof system with polynomial-size proofs of non-isomorphism. Similar to the Cook–Reckhow program [15] for the unsatisfiability problem UNSAT, this defines a clear line of research trying to provide superpolynomial size lower bounds for refuting graph (non)isomorphism formulas in stronger and stronger proof systems. The situation is even more interesting here than in the SAT case since it was proven in [4] that GI is in co-AM, a randomized version of co-NP. Hence, it would not be too surprising if GI $\in$ co-NP, and this would imply the existence of polynomial-size proofs for the problem in some system.

In a recent line of work, the power of different proof systems has been studied with respect to their power in refuting graph isomorphism. The first example of such a lower bound was given in [51] for the Resolution proof system. This result led to lower bounds for stronger proof systems. These studies also make use of the Weisfeiler–Leman algorithm. The authors of [6] exactly characterized the power of the Weisfeiler–Leman algorithm in terms of an algebraic proof system between degree-k Nullstellensatz and degree-k Polynomial Calculus. Moreover, it has been shown in [1, 42, 26] that the power of k-WL lies between the k-th and $(k+1)$-st level of the canonical Sherali–Adams LP hierarchy [49]. Furthermore, it was shown in [44] and independently in [14] that pairs of non-isomorphic n-vertex graphs exist such that any Sum-of-Squares proof of non-isomorphism must have degree $\Omega(n)$. Closely related are the results of [2] that show that Sum-of-Squares degree and Polynomial Calculus degree correlate to the Weisfeiler–Leman dimension (up to constant factors). Recently, in [52], an exact connection was shown between the width and depth measures in (narrow) Resolution and the number of variables and the quantifier depth needed to distinguish a pair of graphs by first-order logic sentences. This result extends to a lower bound for the strong SRC-1 proof system, equipping Resolution with a symmetry rule [53].

1.1 Our Results and Techniques

We show a strong connection between the Weisfeiler–Leman graph differentiation number and the geometric Cutting Planes proof system. We write $G \equiv_k^{\mathrm{CP}} H$ if there is no width-k Cutting Planes refutation of $\mathrm{Iso}(G, H)$, the set of inequalities encoding the statement that the graphs G and H are isomorphic. Further, we write $G \equiv_k^{\mathrm{WL}} H$ if the graphs G and H cannot be distinguished using the logic $\mathcal{C}^k$. Our main result is the following theorem.

▶ **Theorem 1** (Main Result). *Let G and H be two non-isomorphic graphs. Then,*

$$G \equiv_k^{\mathrm{CP}} H \implies G \equiv_k^{\mathrm{WL}} H \implies G \equiv_{k-2}^{\mathrm{CP}} H. \tag{1}$$

In other words,

1. *If $\mathsf{WL}(G, H) \leq k$, then $\mathrm{Iso}(G, H)$ can be refuted by Cutting Planes using width k.*
2. *If $\mathsf{WL}(G, H) > k$, then $\mathrm{Iso}(G, H)$ is not refutable in Cutting Planes using width $k - 2$.*

We achieve the first result by using the winning positions of Spoiler in the bijective pebble game to derive the necessary inequalities. The second result is shown by constructing a set of matrices that "protect" a given point in the isomorphism polytope from being cut away using cuts of a certain width. This result is achieved by proving a so-called protection lemma for graph isomorphism. This type of lemmata has a long tradition in combinatorial optimization (see, e. g., [36]) and has also been used in the area of proof complexity in [10, 18, 40]. The concrete matrices are being constructed using winning positions for Duplicator in the bijective pebble game. From the first result, we can derive polynomial-size CP refutations for isomorphism formulas for graphs with constant WL-dimension.

We also show a size lower bound for refuting graph isomorphism formulas in the subsystem of tree-like Cutting Planes with polynomially bounded coefficients by using known results from communication complexity.

1.2 Organization of This Paper

The remainder of this paper is organized as follows. Section 2 introduces our notation, the Cutting Planes proof system, the Gomory–Chvátal rule, our encoding of graph isomorphism as a set of affine inequalities, and necessary tools from descriptive complexity. We proceed in Section 3 by showing the tight connection between the Weisfeiler–Leman differentiation number for graphs and the width of refuting the corresponding graph isomorphism formulas in the Cutting Planes proof system. Section 4 establishes the lower bound for isomorphism formulas in Tree-CP with polynomially bounded coefficients. Due to space constraints, the proofs of some lemmas are presented in the full-length version of the paper.

2 Preliminaries

2.1 Notation

We let $\mathbb{N}$ denote the set of positive integers, and for $n \in \mathbb{N}$, we define $[n] := \{k \in \mathbb{N} \mid 1 \leq k \leq n\}$. This paper will denote tuples, vectors, and matrices in boldface. Given two vectors $\mathbf{x}, \mathbf{a} \in \mathbb{R}^n$, we let $\langle \mathbf{a}, \mathbf{x} \rangle := \sum_{i=1}^n a_i x_i$ denote the standard inner product.

2.2 The Cutting Planes Proof System

In this paper, we consider Cutting Planes as an inference system used for refuting unsatisfiable CNF formulas, as suggested by [16]. For this, a CNF formula F is translated into a system of affine inequalities that have a 0-1-solution if and only if the corresponding assignment satisfies F. These inequalities can then be manipulated according to certain rules. It is known that a formula is unsatisfiable if and only if, applying these rules, it is possible to obtain the contradiction $0 \geq 1$. A clause $C = (\ell_1 \vee \cdots \vee \ell_k)$ is converted to $\tau(C) \equiv \left[\tau(\ell_1) + \cdots + \tau(\ell_k) \geq 1\right]$, where for each literal ℓ_i, we let $\tau(\ell_i) := x$ if $\ell_i = x$ and $\tau(\ell_i) := 1 - x$ if $\ell_i = \neg x$. We also add the additional inequalities $x \geq 0$ and $-x \geq -1$ for each variable x, forcing them to take values between 0 and 1 (this is a relaxation of the condition $x \in \{0, 1\}$).

▶ **Definition 2.** *Let $\mathbf{a} \in \mathbb{Z}^n$, $\mathbf{a^i} \in \mathbb{Z}^n, \gamma_i \in \mathbb{Z}$ for $i \in [m]$, and $\mathbf{x}$ be a vector of n variables. The Cutting Planes proof system* (CP) *has two rules:*

Linear combination: *From the linear inequalities $\langle \mathbf{a^1}, \mathbf{x} \rangle \geq \gamma_1, \ldots, \langle \mathbf{a^m}, \mathbf{x} \rangle \geq \gamma_m$ and non-negative integers $\alpha_1, \ldots, \alpha_m$, we can derive the inequality $\sum_{i=1}^{m} \alpha_i \langle \mathbf{a^i}, \mathbf{x} \rangle \geq \sum_{i=1}^{m} \alpha_i \gamma_i$.*

Rounding: *From $\langle \mathbf{a}, \mathbf{x} \rangle \geq \gamma$, if all the coefficients in $\mathbf{a}$ are divisible by a positive integer $b > 0$, then we can derive the inequality $\langle \frac{\mathbf{a}}{b}, \mathbf{x} \rangle \geq \lceil \frac{\gamma}{b} \rceil$.*

We can assume, without loss of generality, as done in [10], that a rounding operation is always applied after each application of the linear combination rule and, therefore, both rules can be merged into a single one (called *Gomory–Chvátal cut*, *GC cut* in [10]).

▶ Remark 3. As is standard (see, e. g., [36]), we will sometimes write $a \leq b$ or $-b \leq -a$ for a Cutting Planes inequality of the form $b \geq a$ when it is more natural in our arguments.

▶ **Definition 4.** *A* Cutting Planes refutation *for a set of affine inequalities $f = \{f_1, \ldots, f_m\}$, is a sequence $(g_1, \ldots, g_t)$ of affine inequalities satisfying that*

- *each g_i is either an inequality in f (an* axiom*) or is obtained from previous inequalities by a GC cut,*
- *and g_t is the inequality $0 \geq 1$.*

It is well-known that all the above-mentioned derivation rules are sound for integer solutions. Furthermore, the proof system is complete in the sense that each unsatisfiable CNF formula has a Cutting Planes refutation (see, e. g., [12]).

A CP refutation can be represented in the usual way as a directed acyclic graph in which each vertex corresponds to an affine inequality in the proof. The axioms are the sources, the inequality $0 \geq 1$ is the only sink, and for every application of a GC cut, there is an edge pointing from each of the vertices whose corresponding inequalities are involved in the cut to the vertex representing the result of the cut. The most common complexity measure for a CP refutation is its *size*, defined as the number of vertices in the refutation graph. Two other complexity measures play a central role in our results:

▶ **Definition 5.** *The* rank *of a* CP *refutation (also called* depth*) is the length of the longest path from an axiom to the $0 \geq 1$ inequality in the refutation graph.*

The cutwidth*, or just* width*, of a* CP *refutation is the maximum number of variables in an inequality that results from a GC cut. By this, we mean the number of variables remaining after the linear combination in the rule has been performed or, equivalently, the number of variables after the GC cut (linear combination plus rounding) has been done. If no GC cut is used, we consider the cutwidth to be 0.*

For any complexity measure $\mathcal{C}$ and any unsatisfiable system of affine inequalities f, the $\mathcal{C}$-complexity *of f is the minimum value of $\mathcal{C}$ over all* CP *refutations of f.*

2.3 Two Sets of Affine Inequalities for Graph Isomorphism

We only deal *undirected simple graphs*. Such a graph is a tuple $G = (V_G, E_G)$, where V_G is a finite set of *vertices* and $E_G \subseteq \binom{V_G}{2}$ is the set of *edges*. For a vertex v in a graph G, we denote by $N_G(v)$ the set of its neighbors, and for a set of vertices S, we define $N_G(S)$ as the set of neighbors of the vertices in S. If the graph is clear from the context, we drop the subscripts.

Two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are *isomorphic* if there is a bijection $\varphi\colon V_G \to V_H$ (called *isomorphism from G to H*) such that $\{u, v\} \in E_G \iff \{\varphi(u), \varphi(v)\} \in E_H$ holds for all $u, v \in V_G$. We will denote this by $G \cong H$.

Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs with $V_G = V_H = \{1, \dots, n\}$. We will use the set of variables $x_{i,j}$ with $i, j \in [n]$. If $x_{i,j}$ is greater than 0, this indicates that vertex i in G is mapped to vertex j in H.

For convenience, we consider two different sets of inequalities for which there is a satisfying integer assignment if and only if there is an isomorphism between G and H. The first set of affine inequalities is the one usually used in linear optimization. Let $\mathbf{A}$ and $\mathbf{B}$ be the adjacency matrices of the graphs G and H. The graphs are isomorphic if and only if there is a permutation matrix $\mathbf{X}$ satisfying $\mathbf{AX} = \mathbf{XB}$. This is expressed by the following sets of inequalities. To keep the following definition concise, we write two inequalities $a \leq b$ and $b \leq a$ as the equality $a = b$.

▶ **Definition 6** (MIso Formulas). *The set of affine inequalities* $\mathrm{MIso}(G, H)$ (*for matrix isomorphism*) *contains the following axioms:*

Type 1 axioms: *For every $v \in V_G$ the equality $\sum_{w \in V_H} x_{v,w} = 1$; and for every $w \in V_H$ the equality $\sum_{v \in V_G} x_{v,w} = 1$. Applied to the matrix $\mathbf{X}$, these axioms mean that the sum of each row, as well as the sum of each column, is one.*

Type 2 axioms: *These encode the matrix product $\mathbf{AX} = \mathbf{XB}$. For each position $(i, j) \in [n]^2$, we have the equality $(\mathbf{AX})_{i,j} = (\mathbf{XB})_{i,j}$, or alternatively $\sum_{k \in N(i)} x_{k,j} = \sum_{\ell \in N(j)} x_{i,\ell}$.*

Type 3 axioms: *These are for every variable x the CP axioms $x \leq 1$ and $x \geq 0$.*

An alternative set of affine inequalities over the same set of variables is sometimes more convenient and has been used before for encoding the isomorphism principle in other proof systems like Resolution [51, 48, 52] or Polynomial Calculus [6]. Instead of the inequalities for the matrices, for every two pairs of vertices $v, v' \in V_G$ and $w, w' \in V_H$ such that (v, v') is an edge in G and (w, w') is not an edge in H (or the other way around) we include an inequality indicating that v is not mapped to w or v' is not mapped to w'.

▶ **Definition 7** (Iso Formulas). *The set* $\mathrm{Iso}(G, H)$ *contains the following inequalities:*

Type 1 and Type 3 axioms: *These are exactly the same as in the* MIso *formulas.*

Type 2 axioms: *For every $v, v' \in V_G$ and $w, w' \in V_H$ such that $\big\{(v, w), (v', w')\big\}$ is not an isomorphism in the graphs induced by $\{v, v'\}$ and $\{w, w'\}$, the inequality $x_{v,w} + x_{v',w'} \leq 1$, indicating that an edge cannot be mapped to a non-edge or vice-versa.*

Both systems of inequalities have the same set of 0-1 solutions, which encode the isomorphisms between G and H but can have different sets of fractional solutions. For example, setting all variables $x_{i,j}$ to $\frac{1}{n}$ is always a solution for $\mathrm{Iso}(G, H)$ (even when the graphs are non-isomorphic) but only satisfies $\mathrm{MIso}(G, H)$ when these are regular graphs. A fractional isomorphism is a solution that satisfies the MIso formulas but is not necessarily integral.

▶ **Definition 8.** *The graphs G and H are* fractional isomorphic *if there exists a doubly stochastic matrix $\mathbf{P}$ with $\mathbf{AP} = \mathbf{PB}$, where $\mathbf{A}$ and $\mathbf{B}$ are the adjacency matrices of G and H, respectively. The matrix $\mathbf{P}$ is then called* fractional isomorphism *between G and H.*

We show that there are short CP derivations of each set of inequalities from the other, although, for the derivation of $\mathrm{MIso}(G, H)$ from $\mathrm{Iso}(G, H)$, we need to use the CG-cut rule.

▶ **Lemma 9.** *There is a polynomial-size* CP *derivation of the set of inequalities* $\mathrm{Iso}(G, H)$ *from* $\mathrm{MIso}(G, H)$ *without using the GC cut rule.*

Since fractional solutions can only be eliminated in CP using the GC cut rule, this result implies that the set of solutions of $\mathrm{MIso}(G, H)$ is included in the set of solutions of $\mathrm{Iso}(G, H)$. We consider a derivation in the other direction:

▶ **Lemma 10.** *For any two connected graphs G, H with maximum degree d, there is a polynomial-size CP derivation with rank 2 and width $2d$ of the set of inequalities* $\mathrm{MIso}(G, H)$ *from* $\mathrm{Iso}(G, H)$*.*

2.4 The Weisfeiler–Leman Number and the Bijective k-Pebble Game

In order to express different properties of graphs by certain fragments of first-order logic sentences, Immerman introduced the following definition.

▶ **Definition 11** ([32, 33])**.** *For a logic $\mathcal{L}$ (of first-order logic sentences), the graphs G and H are* $\mathcal{L}$-equivalent*, denoted by $G \equiv_{\mathcal{L}} H$, if for all sentences $\psi \in \mathcal{L}$ it holds that*

$$G \vDash \psi \iff H \vDash \psi.$$

Otherwise, we say that $\mathcal{L}$ can distinguish *G from H, denoted by $G \not\equiv_{\mathcal{L}} H$.*

For $n \in \mathbb{N}$, we introduce a *counting quantifier* $\exists^{\geq n}$. The formula $\exists^{\geq n} x\, \psi$ has the meaning that "there are at least n distinct x satisfying ψ". We also need the notion of quantifier depth (also called quantifier rank).

▶ **Definition 12** ([41])**.** *The* quantifier depth *of a formula ψ is defined inductively as follows:*

- *If ψ is atomic, then $\mathrm{qd}(\psi) = 0$;*
- $\mathrm{qd}(\neg\psi) = \mathrm{qd}(\psi)$*;*
- $\mathrm{qd}(\psi_1 \vee \psi_2) = \max\left\{ \mathrm{qd}(\psi_1), \mathrm{qd}(\psi_2)\right\}$*;*
- $\mathrm{qd}(\exists^{\geq n} x\, \psi) = \mathrm{qd}(\psi) + 1$*.*

▶ **Definition 13.** *The* k-variable counting logic $\mathcal{C}^k$ *is the set of first-order logic formulas that use counting quantifiers but at most k different variables (possibly re-quantifying them). Further, $\mathcal{C}^k_r$ is the subclass of $\mathcal{C}^k$ where the quantifier depth in the formulas is restricted to r.*

For example, $\exists x \left[\exists^{\geq 8} y\, E(x, y) \wedge \forall y \big(E(x, y) \to \exists^{\geq 2} x\, E(y, x)\big)\right]$ lies in $\mathcal{C}^2_3$ and says that there is a vertex that has at least 8 neighbors, each of which has at least 2 neighbors themselves.

▶ **Definition 14.** *The* Weisfeiler–Leman differentiation number *of two graphs G and H is defined by*

$$\mathsf{WL}(G, H) := \begin{cases} \min\{k \in \mathbb{N} \mid G \not\equiv_{\mathcal{C}^k} H\} & \text{if } G \ncong H \\ \infty & \text{if } G \cong H. \end{cases}$$

For a graph G, we say that it has Weisfeiler–Leman dimension *at most k if and only if $G \not\equiv_{\mathcal{C}^{k+1}} H$ for all graphs H non-isomorphic to G.*

Let G and H be two graphs for the remainder of this section. We describe the r-round bijective k-pebble game of Hella [29], adapting the excellent notation from [1]. This game can be used to test $\mathcal{C}^k_r$-equivalence. We first describe some notation and the concept of partial isomorphism before proceeding to introduce the game itself.

▶ **Notation 15.** Let $k \in \mathbb{N}$. Suppose $\mathbf{v} = (v_1, \dots, v_k) \in (V_G \cup \{\star\})^k$. For $i \in [k]$ and $v \in V_G \cup \{\star\}$, we let $\mathbf{v}[i/v]$ denote the tuple $(v_1, \dots, v_{i-1}, v, v_{i+1}, \dots, v_k)$. Further, we let $|\mathbf{v}|_\star$ denote the *number of stars* in the tuple $\mathbf{v}$.

▶ **Definition 16.** *Let $k \in \mathbb{N}$ and let $\mathbf{v} = (v_1, \dots, v_k) \in (V_G \cup \{\star\})^k$ and $\mathbf{w} = (w_1, \dots, w_k) \in (V_H \cup \{\star\})^k$ be two k-tuples. We say that the pair $(\mathbf{v}, \mathbf{w})$* induces/is a partial isomorphism *between G and H if, for every $i, j \in [k]$ we have:*

1. *$v_i = \star$ if and only if $w_i = \star$;*
2. *$v_i = v_j$ if and only if $w_i = w_j$;*
3. *$\{v_i, v_j\} \in E_G$ if and only if $\{w_i, w_j\} \in E_H$.*

In the following game, Spoiler wants to exhibit a difference between the given graphs, while Duplicator tries to disguise such a difference by maintaining a partial isomorphism.

▶ **Definition 17.** *Let $k, r \in \mathbb{N}$. The* r-round bijective k-pebble game *on the graphs G and H is played by two players, called* Spoiler *and* Duplicator. *There are k pairs of matched pebbles in the game. The game proceeds in rounds. The game position after round r is finished can be represented by a pair $(\mathbf{v}, \mathbf{w}) \in (V_G \cup \{\star\})^k \times (V_H \cup \{\star\})^k$. The game starts with some initial position $(\mathbf{v^0}, \mathbf{w^0})$. If this initial tuple does not induce a partial isomorphism between the graphs, Spoiler wins the game after 0 rounds. We now describe the round $r + 1$ of the game. For this, we suppose that the position after round r is given by $(\mathbf{v}, \mathbf{w})$.*

- *If $|\mathbf{v}|_\star = |\mathbf{w}|_\star = 0$, Spoiler must choose a position $i \in [k]$ (otherwise, he can still opt to do this deletion step). The tuples are updated to $\mathbf{v}[i/\star]$ and $\mathbf{w}[i/\star]$.*
- *Duplicator then chooses a bijection $\varphi\colon V_G \to V_H$ (if no such bijection exists, she has lost).*
- *Spoiler picks a vertex $v \in V_G$ and a position $i \in [k]$ such that $v_i = w_i = \star$, and the tuples are updated to $\mathbf{v}[i/v]$ and $\mathbf{w}[i/\varphi(v)]$.*

If the new $(\mathbf{v}, \mathbf{w})$ does not induce a local isomorphism, then Spoiler has won after $r + 1$ rounds. *Otherwise, the game continues with the next round. We say that* Duplicator has a winning strategy *if she can make the game last indefinitely.*

It was shown in [11, 29] that $\mathsf{WL}(G, H) \leq k$ if and only if Spoiler has a winning strategy for the bijective k-pebble game on G and H starting from the initial position $(\mathbf{v}, \mathbf{w})$ with $\mathbf{v} = \mathbf{w} = (\star, \dots, \star)$.

3 CP Refutations for Isomorphism Formulas

We fix two graphs G and H. For the remainder of this paper, it is sometimes convenient to use an alternative view of the pebbling configurations used in Section 2.4.

▶ **Definition 18** (zip Operator)**.** *Let $k \in \mathbb{N}$ and let $\mathbf{v} = (v_1, \dots, v_k) \in (V_G \cup \{\star\})^k$ and $\mathbf{w} = (w_1, \dots, w_k) \in (V_H \cup \{\star\})^k$. We write*

$$p = \mathrm{zip}(\mathbf{v}, \mathbf{w})$$

to denote the set $p \subseteq V_G \times V_H$ given by

$$p := \big\{(v_i, w_i) \bigm| i \in [k] \text{ such that } v_i \neq \star \text{ and } w_i \neq \star\big\}.$$

Definition 16 can easily be adapted to game positions denoted in the way above.

▶ **Notation 19.** Let $k, r \in \mathbb{N}$. For a game position $p \subseteq V_G \times V_H$, we write $p \in D^k(G, H)$ if p is a winning position for Duplicator in the k-bijective game played on G and H. Similarly, the set $D^k_r(G, H)$ is defined for the positions in which Duplicator does not lose in r rounds in the game. We use the notation $S^k_r(G, H)$ to denote winning positions for Spoiler in the respective game.

As in [1, 6], we now define an equivalence relation on $(V_G \cup \{\star\})^k \cup (V_H \cup \{\star\})^k$.

▶ **Definition 20** ([1, 6])**.** *Let $k \in \mathbb{N}$. Further, let G and H be two graphs and let $K, K' \in \{G, H\}$, not necessarily distinct. Additionally, let $\mathbf{u} \in (V_K \cup \{\star\})^k$ and $\mathbf{u}' \in (V_{K'} \cup \{\star\})^k$. We write $\mathbf{u} \equiv_{D^k} \mathbf{u}'$ if $p := \mathrm{zip}(\mathbf{u}, \mathbf{u}') \in D^k(K, K')$.*

It was shown in [1, Lemma 3] that $\equiv_{D^k}$ is an equivalence relation.

3.1 Constructing a CP Refutation from the Bijective Pebble Game

We show that if a pair (G, H) of non-isomorphic graphs can be separated by the bijective k-pebble game in r rounds, then there is a CP refutation for $\mathrm{Iso}(G, H)$ having width k and rank r simultaneously. By Lemma 9, the same result holds for the $\mathrm{MIso}(G, H)$ formulas. We use the equivalence relation $\equiv_{D^k}$ to define a bipartite graph with certain properties.

▶ **Definition 21.** *Let $p \subseteq V_G \times V_H$ be an initial position of the bijective pebble game played on the graphs G and H. The bipartite graph $B^k_r(p)$ is defined as $B := B^k_r(p) = (V_G \uplus V_H, E_B)$ with edge set*

$$E_B := \left\{\{v, w\} \,\middle|\, p \cup \left\{(v, w)\right\} \notin S^k_r(G, H)\right\}.$$

We need the following result from [6]:

▶ **Lemma 22.** *Suppose that Spoiler has a winning position for the bijective k-pebble game played on the graphs G and H in $r{+}1$ rounds starting from position p. In the graph $B := B^k_r(p)$ there are two sets $S \subseteq V_G$ and $T \subseteq V_H$ with the following properties:*

- *$N(S) = T$, $N(T) = S$, and $|S| > |T|$;*
- *Spoiler can win the game in r rounds from the starting position $p \cup \left\{(v, w)\right\}$ for every pair $(v, w) \in V_G \times V_H$ with the property $v \in S \leftrightarrow w \notin T$.*

Proof. By assumption, $p \in S^k_{r+1}(G, H)$. This means that for all bijections $\varphi \colon V_G \to V_H$ that Duplicator can provide, there is always a $v \in V_G$ that Spoiler can choose in return, such that he still has a winning strategy from the position $p \cup \left\{v, \varphi(v)\right\}$ in r rounds. Hence for this v, we have $\left\{v, \varphi(v)\right\} \notin E_B$. Thus, there can be no perfect matching in the graph B. By Hall's marriage theorem [28], a set $S \subseteq V_G$ exists with $|N_B(S)| < |S|$. We choose S to be an inclusion-maximal set with this property. Further, let

$$T := N_B(S).$$

We claim that $N_B(T) = S$ holds. To reach a contradiction, suppose that there is a vertex

$$v \in N_B(T) \setminus S. \tag{2}$$

The maximality of S implies $N_B(v) \not\subseteq T$. Let

$$w \in N_B(v) \setminus T \tag{3}$$

be a vertex witnessing this fact. Since $v \in N_B(T)$, there exists a vertex

$$w' \in N_B(v) \cap T. \tag{4}$$

Moreover, since $T = N_B(S)$ there is a vertex

$$v' \in N_B(w') \cap S. \tag{5}$$

The choice of the vertices in Equations (2) – (5) implies that there are edges

$$\{v', w'\}, \{v, w'\}, \{v, w\} \in E_B.$$

This means that Duplicator has a winning strategy for the r-round k-bijective pebble game from the starting positions $p \cup \big\{(v', w')\big\}$, $p \cup \big\{(v, w')\big\}$, and $p \cup \big\{(v, w)\big\}$. Since $\equiv_{D^k}$ is an equivalence relation, we thus have that she also has a winning strategy starting from $p \cup \big\{(v', w)\big\}$. Hence, $\{v', w\} \in E_B$. However, this contradicts $w \notin N_B(S)$. ◀

For a game position $p = \big\{(v_1, w_1), \ldots, (v_\ell, w_\ell)\big\} \subseteq V_G \times V_H$ we let $S_p := \sum_{i=1}^{\ell} x_{v_i,w_i}$. Note that, in particular, $S_\emptyset = 0$.

▶ **Theorem 23.** *Suppose that Spoiler has a winning strategy for the r-round bijective k-pebble game played on the graphs G and H with initial position p_0. Then, there is a* CP *derivation of the inequality $S_{p_0} \leq |p_0| - 1$ from* Iso(G, H) *having width k and rank r simultaneously.*

Proof. We prove the theorem by induction on r, the number of rounds in the game. First, we consider the base case, where Spoiler wins the game from p_0 in 0 rounds. Since $|V_G| = |V_H|$, it must be that p_0 is not a local isomorphism; therefore, there are two pairs $(v, w), (v', w') \in p_0$ that induce a local non-isomorphism. Hence, the inequality $x_{v,w} + x_{v',w'} \leq 1$ must be a Type 2 axiom of Iso(G, H). Adding the Type 3 axiom inequalities $x_{a,b} \leq 1$ for all $|p_0| - 2$ many other pairs $(a, b) \in p_0 \setminus \big\{(v, w), (v', w')\big\}$, we obtain a derivation of $S_{p_0} \leq |p_0| - 1$.

For the induction step, let $p \subseteq p_0$ with $|p| = \ell < k$ be the set of pairs not deleted by Spoiler at the beginning of the first round in the game. It suffices to show that it is possible to derive the inequality $S_p \leq |p| - 1$. We consider the bipartite graph B from Definition 21. From Lemma 22, we know that there are two sets $S \subseteq V_G$ and $T \subseteq V_H$ with $N(S) = T$, $N(T) = S$ and $|S| > |T|$ and such that for every pair (v, w) with $v \in S \leftrightarrow w \notin T$, Spoiler can win the game in r-rounds from the start position $p \cup \big\{(v, w)\big\}$. By the induction hypothesis, there is a CP derivation of $S_p + x_{v,w} \leq |p|$ for all such pairs.

We notice first that we can derive the inequalities

$$\sum_{v \in S,\, w \in T} x_{v,w} \leq |T| \quad \text{and} \quad \sum_{v \in \overline{S},\, w \in \overline{T}} x_{v,w} \leq |\overline{S}|. \tag{6}$$

To derive the first one of them, observe that for each $w \in T$, we have the axiom inequality $\sum_{v \in V_G} x_{v,w} \leq 1$, which can be reduced to $\sum_{v \in S} x_{v,w} \leq 1$ by adding the axioms $x_{v,w} \geq 0$ for all $v \in \overline{S}$. We obtain the first expression by adding the inequalities for all $w \in T$. The second one is completely analogous. Adding both inequalities of (6) together, we get:

$$\sum_{v \in S,\, w \in T} x_{v,w} + \sum_{v \in \overline{S},\, w \in \overline{T}} x_{v,w} \leq |T| + |\overline{S}|. \tag{7}$$

Since $|T| + |\overline{S}| < |S| + |\overline{S}| = n$, Inequality (7) can be weakened to

$$\sum_{v \in S,\, w \in T} x_{v,w} + \sum_{v \in \overline{S},\, w \in \overline{T}} x_{v,w} \leq n - 1.$$

Next, for each vertex $v \in S$ adding over all inequalities $S_p + x_{v,w} \le |p|$ corresponding to pairs (v, w) for $w \in \overline{T}$ (derived inductively) we get

$$\sum_{w \in \overline{T}} (S_p + x_{v,w}) \le |\overline{T}|\ell,$$

and adding over all $v \in S$, we obtain

$$\sum_{v \in S} \sum_{w \in \overline{T}} (S_p + x_{v,w}) \le |S||\overline{T}|\ell. \tag{8}$$

Analogously, considering the pairs with $v \notin S$ and $w \in T$, we get

$$\sum_{w \in T} \sum_{v \in \overline{S}} (S_p + x_{v,w}) \le |\overline{S}||T|\ell. \tag{9}$$

Let $\gamma := |S||\overline{T}| + |\overline{S}||T|$. By adding the inequalities corresponding to the long Type 1 axioms for all vertices $v \in V_G$, we can derive the inequality

$$\sum_{v \in V_G,\, w \in V_H} x_{v,w} \ge n.$$

Subtracting (8) and (9) from this, we get

$$\sum_{v \in S,\, w \in T} x_{v,w} + \sum_{v \in \overline{S},\, w \in \overline{T}} x_{v,w} - \gamma S_p \ge n - \gamma\ell.$$

Also subtracting the weakened version of (7), we derive

$$-\gamma S_p \ge 1 - \gamma\ell.$$

Observe that this last inequality has been obtained as the linear combination of axioms and previous inequalities, and therefore, the derivation can be done in one step. Using the rounding rule dividing by γ, we get

$$-S_p \ge \left\lceil \frac{1 - \gamma\ell}{\gamma} \right\rceil = 1 - \ell,$$

which is equivalent to $S_p \le \ell - 1$. The linear combination and the rounding rule count as one use of the GC-rule. ◀

▶ **Corollary 24.** *If $G \not\equiv_{\mathcal{C}_r^k} H$, then there is a* CP *refutation for* Iso(G, H) *having width k and rank r simultaneously.*

Proof. Spoiler can win the game starting at the empty initial position $p_0 = \emptyset$. The above result implies that the contradiction $0 \le -1$ can be derived with the desired parameters. ◀

▶ **Corollary 25.** *If a pair of non-isomorphic graphs G, H with n vertices each can be separated by the bijective k-pebble game, then there is a* CP *refutation for* Iso(G, H) *having size $n^{\mathrm{O}(k)}$.*

Proof. This follows from the observation that the CP refutation of Iso(G, H) described above only contains axioms and inequalities of the form $S_p \le |p| - 1$ for sets of pairs p. Since there are at most $\sum_{i=0}^{k} \binom{n}{i}^2 = n^{\mathrm{O}(k)}$ such sets of pairs, the result follows. ◀

Grohe [24] proved that two non-isomorphic graphs in every non-trivial minor-closed graph class can be distinguished using k-WL for a constant k. This implies that for these graphs, the Cutting Planes procedure can produce polynomial-size certificates of graph non-isomorphism. As a concrete example, we mention that it was shown in [38] that the Weisfeiler–Leman dimension of the class of all finite planar graphs is at most 3. Furthermore, 2-WL asymptotically almost surely decides isomorphism for random regular graphs [8, 39].

3.2 CP Width Lower Bound for the Isomorphism Formulas

As described in [36], for a polytope $P \subset \mathbb{R}^n$, the *Chvátal closure* P' is the polytope of all points $\mathbf{x}$ such that, for every $\mathbf{a} \in \mathbb{Z}^n$ and every $b \in \mathbb{R}$, we have

$$\big[\forall \mathbf{y} \in P: \ \langle \mathbf{a}, \mathbf{y} \rangle \geq b\big] \implies \langle \mathbf{a}, \mathbf{x} \rangle \geq \lceil b \rceil; \tag{10}$$

that is, we remove all points of the polytope P that are (in a certain sense) definitely not integer solutions. By iteratively defining $P^{(i+1)} := (P^{(i)})'$, we obtain a sequence $P = P^{(0)} \supseteq P^{(1)} \supseteq P^{(2)} \supseteq \ldots$ of polytopes. The Chvátal rank can then be seen as the smallest r such that $P^{(r)} = P^{\mathbb{Z}}$ (it was shown by Schrijver [47] that such an r always exists).

Protection lemmas have a long tradition in optimization theory for the study of the Chvátal rank. For the CP rank, such lemmas have been used in [10] and [40]. A protection lemma for CP width was introduced in [18]. We give a width protection lemma adapted to the graph isomorphism problem. This generalizes (10). The following notation is employed.

▶ **Notation 26.** Given a matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$ and a set $I \subseteq [n]$ we denote by $\mathbf{X}|_I \in \mathbb{R}^{n \times n}$ the projection of $\mathbf{X}$ to the rows in I, that is, to the positions $\mathrm{Rows}[I] := \{(i, j) \mid i \in I, \, j \in [n]\}$ (meaning that the rows which are not in I are set to 0).

▶ **Definition 27.** *Let G and H be two graphs with n vertices each and let $P_{G,H}$ be the polytope in $[0,1]^{n \times n}$ defined by the* $\mathrm{MIso}(G, H)$ *inequalities. For $k \in \mathbb{N}$, we define*

$$P'_{G,H}(k) := \left\{ \mathbf{X} \in P_{G,H} \;\middle|\; \begin{array}{l} \forall \mathbf{A} \in \mathbb{Z}^{n \times n}, \ \forall b \in \mathbb{R}, \ \forall I \subseteq [n] \ \textit{with} \ |I| = k: \\ \big[\forall \, \mathbf{Y} \in P_{G,H}: \ \langle \mathbf{A}, \mathbf{Y}|_I \rangle_{\mathrm{F}} \geq b\big] \implies \langle \mathbf{A}, \mathbf{X}|_I \rangle_{\mathrm{F}} \geq \lceil b \rceil \end{array} \right\}.$$

Here $\langle \mathbf{A}, \mathbf{B} \rangle_{\mathrm{F}} := \sum_{i=1}^n \sum_{j=1}^n a_{i,j} \, b_{i,j}$ denotes the Frobenius inner product *between the matrices $\mathbf{A} = (a_{i,j}) \in \mathbb{R}^{n \times n}$ and $\mathbf{B} = (b_{i,j}) \in \mathbb{R}^{n \times n}$.*

▶ **Lemma 28** (Protection Lemma for Graph Isomorphism). *Let $k \in \mathbb{N}$. Further, let $\mathbf{X} \in [0,1]^{n \times n}$ be a fractional isomorphism in the polytope $P_{G,H}$. Suppose that for any $I \subseteq [n]$ with $|I| \leq k$, there is a set of matrices $\mathbf{Y^1}, \ldots, \mathbf{Y^s} \in [0,1]^{n \times n}$ satisfying:*

- *For all $t \in [s]$, $(\mathbf{Y^t})_{i,j} \in \{0, 1\}$ in all positions $(i, j) \in \mathit{Rows}[I]$;*
- *for all $t \in [s]$, the matrix $\mathbf{Y^t}$ is a fractional solution of $P_{G,H}$; and*
- *the restriction $\mathbf{X}|_I$ is a convex combination of $\mathbf{Y^1}|_I, \ldots, \mathbf{Y^s}|_I$.*

Then, $\mathbf{X} \in P'_{G,H}(k)$.

Proof. Suppose, to reach a contradiction, that $\mathbf{X}$ is a fractional isomorphism in $P_{G,H}$ but $\mathbf{X} \notin P'_{G,H}(k)$. Then, by Definition 27 there exists a matrix $\mathbf{A} \in \mathbb{Z}^{n \times n}$, a real number $b \in \mathbb{R}$, and a set $I \subseteq [n]$ with $|I| = k$ such that for all $\mathbf{Y} \in P_{G,H}$ we have $\langle \mathbf{A}, \mathbf{Y}|_I \rangle_{\mathrm{F}} \geq b$ but $\langle \mathbf{A}, \mathbf{X}|_I \rangle_{\mathrm{F}} < \lceil b \rceil$. Since $\mathbf{X} \in P_{G,H}$, we have $\langle \mathbf{A}, \mathbf{X}|_I \rangle_{\mathrm{F}} \geq b$. This implies that $\langle \mathbf{A}, \mathbf{X}|_I \rangle_{\mathrm{F}} \notin \mathbb{Z}$.

For all the protection matrices $\mathbf{Y^t} \in \{\mathbf{Y^1}, \ldots, \mathbf{Y^s}\}$, since they are 0-1-valued in $\mathrm{Rows}[I]$, we have that $\langle \mathbf{A}, \mathbf{Y^t}|_I \rangle_{\mathrm{F}}$ is an integer. Also, since $\mathbf{Y^t}$ is in the polytope, $\langle \mathbf{A}, \mathbf{Y^t}|_I \rangle_{\mathrm{F}} \geq b$. Combining both facts, we have $\langle \mathbf{A}, \mathbf{Y^t}|_I \rangle_{\mathrm{F}} \geq \lceil b \rceil$. However, since $\mathbf{X}|_I$ is a convex combination of the protection matrices, it must hold $\langle \mathbf{A}, \mathbf{X}|_I \rangle_{\mathrm{F}} \geq \lceil b \rceil$, which is a contradiction. ◀

Note that for $|I| = k$, the above restrictions consider kn variables. In previously published protection lemmas, these restrictions had size k. However, our version can only make the construction of the protection matrices harder.

For each game position $p \subseteq V_G \times V_H$, we define a matrix $\mathbf{M^p}$ that we will show in Lemma 35 to be a fractional isomorphism between G and H. We begin by first defining auxiliary functions that will be used to define the entries of this matrix. Since $\equiv_{D^k}$ is an equivalence relation, we can define the type of $\mathbf{v} \in (V_G \cup \{\star\})^k$ as the equivalence class of $\mathbf{v}$.

▶ **Definition 29** ([6]). *Given a tuple $\mathbf{v} \in (V_G \cup \{\star\})^k$ we let*

$$c(\mathbf{v}) := [\mathbf{v}]_{\equiv_{D^k}}$$

be the equivalence class of $\mathbf{v}$. Further, we define

$$t(\mathbf{v}) := |c(\mathbf{v}) \cap (V_G \cup \{\star\})^k|.$$

▶ **Definition 30.** *Let $\mathbf{v} \in (V_G \cup \{\star\})^k$ and $\mathbf{w} \in (V_H \cup \{\star\})^k$. For every non-empty game position $q = \mathrm{zip}(\mathbf{v}, \mathbf{w}) \neq \emptyset$, the function ζ is defined in the following way:*

$$\zeta(q) := \begin{cases} 0 & \text{if } c(\mathbf{v}) \neq c(\mathbf{w}) \\ \frac{1}{t(\mathbf{v})} & \text{otherwise.} \end{cases} \tag{11}$$

For $q = \emptyset$, we let $\zeta(\emptyset) := 1$.

We use the function ζ to define the entries of the matrix. For a game position $p \subseteq V_G \times V_H$ and a tuple $(v, w) \in V_G \times V_H$, we use the notation $p \cup vw$ as a shorthand for $p \cup \{(v, w)\}$.

▶ **Definition 31.** *Let $p \subseteq V_G \times V_H$ be a game position with $|p| \leq k - 1$. For every $i, j \in [n]$, the number $m^p_{i,j}$ is defined in the following way:*

$$m^p_{i,j} := \begin{cases} 0 & \text{if } p \cup v_i w_j \notin D^k(G, H) \\ \frac{\zeta(p \cup v_i w_j)}{\zeta(p)} & \text{otherwise.} \end{cases} \tag{12}$$

We further define the matrix $\mathbf{M^p}$ by letting $(\mathbf{M^p})_{i,j} := m^p_{i,j}$ for each $i, j \in [n]$.

Observe that in the case $\zeta(p) = 0$, the value of $m^p_{i,j}$ is 0 because the first case in (11) implies that $p \cup v_i w_j \notin D^k(G, H)$, ensuring that we do not divide by zero in (12). The following result follows directly from the definition of the matrix entries.

▶ **Lemma 32.** *If $p \in D^k(G, H)$ and $(v_i, w_j) \in p$, then*
(i) *$m^p_{i,j} = 1$, and*
(ii) *$m^p_{i',j} = m^p_{i,j'} = 0$ for $i \neq i'$ and $j \neq j'$.*

▶ **Notation 33.** Let $\mathbf{v} = (v_1, \ldots, v_k) \in (V_G \cup \{\star\})^k$ such that there is an $i \in [k]$ with $v_i = \star$. Such tuples represent positions in the bijective pebble game and are thus closed under permutations when the corresponding tuple in the other graph is permuted with the same permutation; see, e. g., [1, Claim 11]. For $v \in V_G$, we let $\mathbf{v}v$ be the tuple that results by replacing any $\star$ in $\mathbf{v}$ with v.

In the following, we tacitly assume an ordering $v_1 \prec v_2 \prec \cdots \prec v_n$ on the vertices of the graph and often identify a vertex v_i with its number i in this order. Hence, we can now speak of matrix positions (v, w). This helps to keep the following notation clear. The following technical lemma is needed in the next results. It follows from the properties of the equivalence relations defined by the bijective game.

▶ **Lemma 34.** *Let $p \subseteq V_G \times V_H$ with $|p| \leq k - 1$, and $p = \mathrm{zip}(\mathbf{a}, \mathbf{b})$. Then, for every $(v, w) \in V_G \times V_H$, if $c(\mathbf{a}v) = c(\mathbf{b}w)$, then*

$$t(\mathbf{a}v) = t(\mathbf{a}) \cdot \left|\left\{w' \in V_H \mid c(\mathbf{a}v) = c(\mathbf{b}w')\right\}\right|.$$

Proof. Since $c(\mathbf{a}v) = c(\mathbf{b}w)$, we have

$$\left|\left\{w' \in V_H \mid c(\mathbf{a}v) = c(\mathbf{b}w')\right\}\right| = \left|\left\{v' \in V_G \mid c(\mathbf{a}v) = c(\mathbf{a}v')\right\}\right| = \frac{|c(\mathbf{a}v)|}{|c(\mathbf{a})|} = \frac{t(\mathbf{a}v)}{t(\mathbf{a})}. \qquad ◀$$

▶ **Lemma 35.** *Let* $\mathbf{a} \in (V_G \cup \{\star\})^k$ *and* $\mathbf{b} \in (V_H \cup \{\star\})^k$ *be such that* $c(\mathbf{a}) = c(\mathbf{b})$ *and such that* $p := \mathrm{zip}(\mathbf{a}, \mathbf{b})$ *has size* $|p| < k - 1$*. Then, the matrix* $\mathbf{M^p}$ *is a fractional isomorphism between* G *and* H*.*

Proof. We first show that when p is as above, the axioms expressed that we are dealing with a double stochastic matrix are satisfied by $\mathbf{M^p}$ (even when $|p| = k-1$). For each $v \in V_G$ we have

$$\sum_{w \in V_H} (\mathbf{M^p})_{v,w} = \sum_{\substack{w \in V_H \\ c(\mathbf{a}\,v) = c(\mathbf{b}\,w)}} \frac{\zeta(p \cup vw)}{\zeta(p)} = \sum_{\substack{w \in V_H \\ c(\mathbf{a}\,v) = c(\mathbf{b}\,w)}} \frac{t(\mathbf{a})}{t(\mathbf{a}\,v)}.$$

By Lemma 34 we have $\big|\{w \in V_H \mid c(\mathbf{a}\,v) = c(\mathbf{b}\,w)\}\big| = t(\mathbf{a}\,v)/t(\mathbf{a})$. Hence, the sum of a row in the matrix adds to 1. The proof for the columns is analogous.

For the case of the isomorphism axioms, let $(v, w) \in V_G \times V_H$. We have to show

$$\sum_{i \in N(v)} m^p_{i,w} = \sum_{j \in N(w)} m^p_{v,j}. \tag{13}$$

By the result on the double stochasticity of the matrices just proved above, since $|p \cup iw| \leq k - 1$, we have that for every $i \in N(v)$,

$$1 = \sum_{j \in V_H} m^{p \cup iw}_{v,j} = \sum_{j \in N(w)} m^{p \cup iw}_{v,j},$$

where the last equality holds because only for the neighbors of w the value of $m^{p\cup iw}_{v,j}$ can be different from 0. By the definition, if $m^{p\cup iw}_{v,j} \neq 0$, then this number can be expressed as $\xi \cdot m^p_{v,j}$, with

$$\xi := \frac{t(\mathbf{a}\,v) \cdot t(\mathbf{a}\,i)}{t(\mathbf{a}) \cdot t(\mathbf{a}\,i\,v)}.$$

Therefore,

$$\sum_{j \in N(w)} m^p_{v,j} = \sum_{j \in N(w)} m^{p \cup iw}_{v,j} \frac{1}{\xi} = \frac{1}{\xi}.$$

Similarly for every $j \in N(w)$, we have

$$1 = \sum_{i \in N(v)} m^{p \cup vj}_{i,w},$$

and also the numbers $m^{p\cup vj}_{i,w}$ (when different from 0) can be expressed as $\xi \cdot m^p_{i,w}$. Therefore both sums in (13) are equal. ◀

We observe that in the previous result, it does not suffice that $|p| \leq k - 1$ in order for the matrix $\mathbf{M^p}$ to be a fractional isomorphism between G and H. As a counterexample consider G to be a cycle with 6 vertices and H to be the union of two cycles with 3 vertices each, and let $\mathbf{A}$ and $\mathbf{B}$ be the adjacency matrices of these graphs. Duplicator wins the 2-pebble game on G, H; however, it can be easily checked that for $p = \{(v, w)\}$ for any pair $v \in V_G, w \in V_H$, the matrix $\mathbf{M^p}$ does not satisfy $\mathbf{AM^p} = \mathbf{M^pB}$.

▶ **Theorem 36.** *Let* G *and* H *be two non-isomorphic graphs with* n *vertices each such that* $G \equiv_{\mathcal{C}^k} H$*. Further, let* $p \in D^k(G, H)$ *with* $|p| < k - 1$ *and consider the matrix* $\mathbf{M^p}$*. For any* $I \subseteq [n]$ *with* $|I| < k - 1$*, there is a set of matrices* $\mathbf{Y^1}, \ldots, \mathbf{Y^s}$*, satisfying:*

- *Each of these matrices is* 0*-*1*-valued on* $\mathit{Rows}[I]$*;*
- *each of these matrices is a fractional isomorphism in* $P_{G,H}$ *of the form* $\mathbf{M}^{\mathbf{p}'}$*, with* $|p'| < k-1$*, and* $p' \in D^k(G,H)$*; and*
- *the restriction* $\mathbf{M}^{\mathbf{p}}|_I$ *is a convex combination of* $\mathbf{Y}^{\mathbf{1}}|_I, \dots, \mathbf{Y}^{\mathbf{s}}|_I$*.*

Proof. We prove the result by induction on $\ell = |I|$. For the induction base $\ell = 1$, let p be a game starting position as above and let $I = \{i_1\}$. Consider the matrix $\mathbf{M}^{\mathbf{p}}$. We can suppose that w. l. o. g. that i_1 is not a vertex contained in a tuple of p and also that for all $j \in [n]$, the matrix entry $m^p_{i_1,j} \neq 1$. Otherwise, $\mathbf{M}^{\mathbf{p}}$ already has the desired properties.

Let us start with the simpler case in which $|p| < k-2$. Also, not all the positions in a row i_1 of $\mathbf{M}^{\mathbf{p}}$ can be 0 since we are dealing with a fractional isomorphism. Under these conditions, there is a set of at least two non-zero elements in that row; let us call this set $\mathrm{NZ}(i_1)$. This follows from the fact that the sum of the row elements adds to 1. For each $j \in \mathrm{NZ}(i_1)$ let $p_j := p \cup v_{i_1} w_j$ and consider the matrices $\mathbf{Y}^{\mathbf{j}} := \mathbf{M}^{\mathbf{p_j}}$ for $j \in \mathrm{NZ}(i_1)$.

These matrices have the following properties: According to Lemma 32, they have 0-1 values on the row i_1. Due to Lemma 35, the matrices $\mathbf{M}^{\mathbf{p_j}}$ are fractional isomorphisms since $|p_j| < k-1$. All the p_j considered as game positions are winning positions for Duplicator in the k-bijective game since they can be reached if Spoiler adds the pair (v_{i_1}, w_j) (for any $j \in \mathrm{NZ}(i_1)$) which is a valid move since these positions are non-zero in $\mathbf{M}^{\mathbf{p}}$, meaning that $p \cup v_{i_1} w_j$ is also a winning position for Duplicator. It is only left to show that $\mathbf{M}^{\mathbf{p}}|_I$ is a convex combination of the restriction to $\mathrm{Rows}[I]$ of the matrices $\mathbf{M}^{\mathbf{p_j}}$, but this follows from the fact that for each $j \in \mathrm{NZ}(i_1)$, the matrix $\mathbf{M}^{\mathbf{p_j}}$ has a 1 in position (i_1, j), and 0's in all other positions in the row i_1, and all these positions have the same value in $\mathbf{M}^{\mathbf{p}}$. Therefore, $\mathbf{M}^{\mathbf{p}}|_I$ can be obtained as a convex combination of the restriction to $\mathrm{Rows}[I]$ of the new matrices, multiplying each one of them times $(\mathbf{M}^{\mathbf{p}})_{i_1,j}$. This is a correct combination since for $p = \mathrm{zip}(\mathbf{a}, \mathbf{b})$, and $c(\mathbf{a}\, v_{i_1}) = c(\mathbf{b}\, w_j)$, we have

$$(\mathbf{M}^{\mathbf{p}})_{i_1,j} = \frac{t(\mathbf{a})}{t(\mathbf{a}\, v_{i_1})}$$

and the number of such matrices is equal to

$$\big|\mathrm{NZ}(i_1)\big| = \big|\{j \mid c(\mathbf{a}\, v_{i_1}) = c(\mathbf{b}\, w_j)\}\big| = \frac{t(\mathbf{a}\, v_{i_1})}{t(\mathbf{a})}.$$

Let us now suppose $|p| = k-2$. In this situation, we cannot just add elements to p since then, we cannot guarantee that the resulting matrix is a fractional isomorphism, and we have to delete some elements from p first. Let (v, w) be any pair in p and let $\hat{p} = p \setminus \big\{(v, w)\big\}$. For any $j \in \mathrm{NZ}(i_1)$ let $p'_j := \hat{p} \cup v_{i_1} w_j$ and consider the matrices $\mathbf{Y}^{\mathbf{j}} := \mathbf{M}^{\mathbf{p'_j}}$. Again these matrices have 0-1 values on the row i_1 and encode fractional isomorphisms since each p'_j has the right length and is a winning position for Duplicator in the k-bijective game since these positions can be reached if Spoiler deletes (v, w) from p and adds (v_{i_1}, w_j), which are valid moves since these positions are non-zero in $\mathbf{M}^{\mathbf{p}}$, meaning that $p \cup v_{i_1} w_j$ is also a winning position for Duplicator and, therefore, p'_j is also one. It is only left to show that $\mathbf{M}^{\mathbf{p}}|_I$ is a convex combination of the restriction to $\mathrm{Rows}[I]$ of the matrices $\mathbf{M}^{\mathbf{p'_j}}$. Let $p = \mathrm{zip}(\mathbf{a}\, v, \mathbf{b}\, w)$. The value of a non-zero position in row i_1 in $\mathbf{M}^{\mathbf{p}}$ is

$$\frac{t(\mathbf{a}\, v)}{t(\mathbf{a}\, v\, v_{i_1})}.$$

If there is a non-zero position in row i_1 in $\mathbf{M}^{\mathbf{p}}$, then the same position in $\mathbf{M}^{\hat{\mathbf{p}}}$ is also non-zero since $\hat{p} \subseteq p$. Each matrix $\mathbf{M}^{\mathbf{p'_j}}$ has a 1 in position (i_1, j) and 0's in the other positions in that row. If $(\mathbf{M}^{\mathbf{p}})_{i_1,j} \neq 0$, then $\mathbf{M}^{\mathbf{p'_j}}$ is one of the $\mathbf{Y}$ matrices since $(\mathbf{M}^{\hat{\mathbf{p}}})_{i_1,j} \neq 0$. There are

$$\left|\{j \mid c(\mathbf{a}\,v\,v_{i_1}) = c(\mathbf{b}\,w\,w_j)\}\right| = \frac{t(\mathbf{a}\,v\,v_{i_1})}{t(\mathbf{a}\,w)}$$

non-zero positions in row i_1 in $\mathbf{M^p}$. Multiplying the $\mathbf{Y}$ matrices corresponding to these positions times $\frac{t(\mathbf{a}\,v)}{t(\mathbf{a}\,v\,v_{i_1})}$ and adding them together, we obtain the convex combination.

The induction step is completely analogous. Given $\mathbf{M^p}$ and $I = \{i_1, \dots, i_\ell\}$, let $I' = \{i_1, \dots, i_{\ell-1}\}$. By induction, we can construct a set of matrices $\mathbf{Y^t}$ of the form $\mathbf{M^{p'}}$ with p' containing a set of pairs $\{(i_1, j_1), \dots, (i_{\ell-1}, j_{\ell-1})\}$ satisfying the conditions and such that $\mathbf{M^p}|_{I'}$ is a convex combination of the constructed matrices. These are 0-1 on Rows$[I']$. In one last step, we can construct from these the matrices for I as in the case for $\ell = 1$. A convex combination from convex combinations is still one. ◀

▶ **Corollary 37.** *If Duplicator has a winning strategy for the k-pebble bijective game played on G, H, then there is no* CP *refutation of* MIso(G, H) *of width $k - 2$.*

Proof. This follows from Lemma 28 and the previous result since they together imply that each $\mathbf{M^p}$ corresponding to a winning position p for Duplicator of size $|p| \le k-2$ survives cuts of size $k - 2$. At each step, starting from the empty position $p = \emptyset$, we consider the fractional isomorphism $\mathbf{M^p}$. There are protection matrices for it that also correspond to winning positions p' for the Duplicator with size $|p'| \le k - 2$. For each of these new positions p' there are also protection matrices and therefore, it is not possible, allowing only cuts of width $k-2$, to eliminate any of these fractional isomorphisms from $P_{G,H}$. ◀

A close inspection of the proof of the previous theorem, together with Lemma 28, also gives a connection to CP rank.

▶ **Corollary 38.** *Let $k \ge 3$. If Duplicator has a winning strategy for the r-round k-pebble bijective game played on G, H, then there is no* CP *refutation of* MIso(G, H) *of width $k - 2$ and rank $\frac{r}{k-2}$.*

4 Tree-CP* Size Lower Bounds for Refuting Isomorphism

Proving size lower bounds for Cutting Planes refutations of the isomorphism problem is a challenging open question. Basically, the two only known methods for proving such bounds are interpolation and lifting. Neither of these methods is suitable for isomorphism formulas. Interpolation requires some monotone problem, and GI is highly non-monotone. Also, after applying lifting, one obtains some constructed formulas that are not isomorphism formulas. Using some known results from communication complexity, we can, however, show size lower bounds for the restricted case of tree-like Cutting Planes proofs with polynomially bounded coefficients (we denote this system with Tree-CP*). A refutation is *tree-like* if the underlying directed acyclic graph is a tree. A CP proof for a formula F has *polynomially bounded coefficients* if there exists a constant $c > 0$ such that the absolute value of all coefficients used in inequalities of the proof is bounded by $\mathrm{O}(n^c)$, where n is the number of variables of F. The system Tree-CP* is non-trivial, allowing, for example, polynomial-sized proofs for the pigeonhole principle [16].

In [34], the size of Tree-CP proofs for a formula F was related to the communication complexity of a search problem for F, showing that if the underlying search problem has high communication complexity, this implies a lower bound for the Tree-CP size of refuting F.

Critical block sensitivity is a communication complexity measure introduced in [31], extending the classical concept of block sensitivity [43]. It is an easy fact that a critical block sensitivity lower bound for a problem implies the same bound for the communication complexity of the search problem.

In [23, Theorem 3], lower bounds on the critical block sensitivity of Tseitin formulas were proved. The authors showed that there exist graph families of bounded degree, with critical block sensitivity communication $\Omega(n/\log n)$ which by the results in [34] imply a size lower bound of $\Omega(2^{n/\log^2 n})$ for Tree-CP* refutations of Tseitin formulas.

It was shown in [50, Lemma 4.2] that there is a direct reduction from Tseitin to isomorphism formulas, and it is, thus, possible to obtain lower bounds for isomorphism formulas from lower bounds for Tseitin formulas. As a direct consequence of all these results, we obtain:

▶ **Corollary 39.** *There are families of non-isomorphic graphs G, H, with n vertices each, and such that the refutation of* $\mathrm{MIso}(G, H)$ *in* Tree-CP* *requires size* $\Omega(2^{n/\log^2 n})$.

5 Conclusions and Open Problems

We have shown an exact characterization of the Weisfeiler–Leman graph differentiation number of two graphs in terms of the cutwidth needed for refuting the corresponding isomorphism formula. Let us emphasize that Equation (1) holds for both the Iso and MIso formulas. For this, we have introduced a new protection lemma for the graph isomorphism polytope. This new connection enabled us to show that the Cutting Planes proof system can show graph non-isomorphism in polynomial time for graphs with a constant Weisfeiler–Leman dimension. Furthermore, by using known results from communication complexity, we were able to give a lower bound for the size of tree-like CP refutations with polynomially bounded coefficients for refuting graph isomorphism inequalities. Some important questions remain open. Maybe the most interesting one is to prove CP size lower bounds for isomorphism formulas. This is quite challenging since basically all the lower bounds for this kind of formula are based on graphs related to the Tseitin formulas, and recently a quasi-polynomial upper bound for the CP size of such formulas has been shown [17]. Furthermore, it would be interesting to have trade-off results between the dimension of the WL algorithm and its iteration number (this is equivalent to a trade-off between the number k of pebbles in Hella's bijective pebble game and the number r of rounds). While trade-off results are known for these parameters [7, 25], they do not hold for structures of bounded arity (like graphs). However, due to the connection of these parameters to the Resolution proof system [52] and the Cutting Planes proof system, as shown in this paper, such results would immediately imply proof complexity trade-offs (in our case, between width and rank for Cutting Planes). Moreover, it is open if the second implication in (1) can be improved.

References

1 Albert Atserias and Elitza N. Maneva. Sherali–Adams relaxations and indistinguishability in counting logics. *SIAM Journal on Computing*, 42(1):112–137, 2013. Earlier conference version in *ITCS '12*. doi:10.1137/120867834.

2 Albert Atserias and Joanna Ochremiak. Definable ellipsoid method, sums-of-squares proofs, and the isomorphism problem. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 18)*, pages 66–75, 2018. doi:10.1145/3209108.3209186.

3 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC '16)*, pages 684–697, 2016. Full-length version in *arXiv:1512.03547*. doi:10.1145/2897518.2897542.

4 László Babai and Shlomo Moran. Arthur–Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988. doi:10.1016/0022-0000(88)90028-1.

5 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001. Earlier conference versions in *STOC '99* and *CCC '99*. doi:10.1145/375827.375835.

6 Christoph Berkholz and Martin Grohe. Limitations of algebraic approaches to graph isomorphism testing. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP '15)*, pages 155–166, 2015. doi:10.1007/978-3-662-47672-7_13.

7 Christoph Berkholz and Jakob Nordström. Near-optimal lower bounds on quantifier depth and Weisfeiler–Leman refinement steps. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16)*, pages 267–276, 2016. doi:10.1145/2933575.2934560.

8 Béla Bollobás. Distinguishing vertices of random graphs. In *North-Holland Mathematics Studies*, volume 62, pages 33–49. Elsevier, 1982. doi:10.1016/S0304-0208(08)73545-X.

9 Maria L. Bonet, Juan L. Esteban, Nicola Galesi, and Jan Johannsen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM Jornal on Computing*, 30(5):1462–1484, 2000. doi:10.1137/S0097539799352474.

10 Joshua Buresh-Oppenheim, Nicola Galesi, Shlomo Hoory, Avner Magen, and Toniann Pitassi. Rank bounds and integrality gaps for cutting planes procedures. *Theory Comput.*, 2(4):65–90, 2006. doi:10.4086/toc.2006.v002a004.

11 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. Earlier conference version in *FOCS '89*. doi:10.1007/BF01305232.

12 Vasek Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(4):305–337, 1973. doi:10.1016/0012-365X(73)90167-2.

13 Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, pages 174–183, May 1996. doi:10.1145/237814.237860.

14 Paolo Codenotti, Grant Schoenebeck, and Aaron Snook. Graph isomorphism and the Lasserre hierarchy, 2014. arXiv:1401.0758.

15 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979. doi:10.2307/2273702.

16 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987. doi:10.1016/0166-218X(87)90039-4.

17 Daniel Dadush and Samarth Tiwari. On the complexity of branching proofs. In *Proceedings of the 35th Computational Complexity Conference (CCC '20)*, volume 169 of *LIPIcs*, pages 34:1–34:35, 2020. doi:10.4230/LIPIcs.CCC.2020.34.

18 Stefan S. Dantchev and Barnaby Martin. Cutting planes and the parameter cutwidth. *Theory of Computing Systems*, 51(1):50–64, 2012. doi:10.1007/s00224-011-9373-0.

19 Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961. doi:10.4064/fm-49-2-129-141.

20 Roland Fraïssé. Sur une nouvelle classification des systèmes de relations. *Comptes Rendus*, 230:1022–1024, 1950.

21 Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. *Theory of Computing*, 16:1–30, 2020. Earlier conference version in *STOC '18*. doi:10.4086/toc.2020.v016a013.

22 Ralph E. Gomory. An algorithm for integer solutions of linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

23 Mika Göös and Toniann Pitassi. Communication lower bounds via critical block sensitivity. *SIAM Journal on Computing*, 47(5):1778–1806, 2018. doi:10.1137/16M1082007.

24 Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic*. Cambridge University Press, 2017. doi:10.1017/9781139028868.

25 Martin Grohe, Moritz Lichter, and Daniel Neuen. The iteration number of the Weisfeiler–Leman algorithm, 2023. arXiv:2301.13317.

26 Martin Grohe and Martin Otto. Pebble games and linear equations. *The Journal of Symbolic Logic*, 80(3):797–844, 2015. Earlier conference version in *CSL '12*. doi:10.1017/jsl.2015.28.

27 Armin Haken and Stephen A. Cook. An exponential lower bound for the size of monotone real circuits. *Journal of Computer and System Sciences*, 58(2):326–335, 1999. doi:10.1006/jcss.1998.1617.

28 Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10(1):26–30, 1935. doi:10.1112/jlms/s1-10.37.26.

29 Lauri Hella. Logical hierarchies in PTIME. *Information and Computation*, 129(1):1–19, 1996. doi:10.1006/inco.1996.0070.

30 Pavel Hrubes and Pavel Pudlák. Random formulas, monotone circuits, and interpolation. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS '17)*, pages 121–131, 2017. doi:10.1109/FOCS.2017.20.

31 Trinh Huynh and Jakob Nordström. On the virtue of succinct proofs: amplifying communication complexity hardness to time-space trade-offs in proof complexity. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC '12)*, pages 233–248, 2012. doi:10.1145/2213977.2214000.

32 Neil Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25(1):76–98, 1982. Earlier conference version in *FOCS '80*. doi:10.1016/0022-0000(82)90011-3.

33 Neil Immerman. *Descriptive Complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

34 Russell Impagliazzo, Toniann Pitassi, and Alasdair Urquhart. Upper and lower bounds for tree-like cutting planes proofs. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science (LICS '94)*, pages 220–228, July 1994. doi:10.1109/LICS.1994.316069.

35 Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, 1999. doi:10.1007/s000370050024.

36 Stasys Jukna. *Boolean Function Complexity – Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

37 Sandra Kiefer. *Power and limits of the Weisfeiler–Leman algorithm*. Dissertation, RWTH Aachen University, 2020. doi:10.18154/RWTH-2020-03508.

38 Sandra Kiefer, Ilia Ponomarenko, and Pascal Schweitzer. The Weisfeiler–Leman dimension of planar graphs is at most 3. *Journal of the ACM*, 66(6):44:1–44:31, 2019. doi:10.1145/3333003.

39 Ludek Kucera. Canonical labeling of regular graphs in linear average time. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science (FOCS '87)*, pages 271–279, 1987. doi:10.1109/SFCS.1987.11.

40 Massimo Lauria. A rank lower bound for cutting planes proofs of Ramsey's theorem. *ACM Transactions on Computation Theory*, 8(4):17:1–17:13, 2016. doi:10.1145/2903266.

41 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science: An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

42 Peter N. Malkin. Sherali–Adams relaxations of graph isomorphism polytopes. *Discrete Optimization*, 12:73–97, 2014. doi:10.1016/j.disopt.2014.01.004.

43 Noam Nisan. CREW PRAMs and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991. doi:10.1137/0220062.

44 Ryan O'Donnell, John Wright, Chenggang Wu, and Yuan Zhou. Hardness of robust graph isomorphism, Lasserre gaps, and asymmetry of random graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14)*, pages 1659–1677, 2014. doi:10.1137/1.9781611973402.120.

45 Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *The Journal of Symbolic Logic*, 62(3):981–998, 1997. doi:10.2307/2275583.

46 Alexander A. Razborov. On the width of semialgebraic proofs and algorithms. *Math. Oper. Res.*, 42(4):1106–1134, 2017. doi:10.1287/moor.2016.0840.

47 Alexander Schrijver. On cutting planes. *Annals of Discrete Mathematics*, 9:291–296, 1980. doi:10.1016/S0167-5060(08)70085-2.

48 Pascal Schweitzer and Constantin Seebach. Resolution with symmetry rule applied to linear equations. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS '21)*, volume 187 of *LIPIcs*, pages 58:1–58:16, 2021. doi:10.4230/LIPIcs.STACS.2021.58.

49 Hanif D. Sherali and Warren P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990. doi:10.1137/0403036.

50 Jacobo Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004. doi:10.1137/S009753970241096X.

51 Jacobo Torán. On the resolution complexity of graph non-isomorphism. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT '13)*, pages 52–66, 2013. doi:10.1007/978-3-642-39071-5_6.

52 Jacobo Torán and Florian Wörz. Number of variables for graph differentiation and the resolution of graph isomorphism formulas. *ACM Transactions on Computational Logic*, 24(3):23:1–23:25, April 2023. Earlier conference version in *CSL '22*. doi:10.1145/3580478.

53 Alasdair Urquhart. The symmetry rule in propositional logic. *Discrete Applied Mathematics*, 96–97:177–193, 1999. doi:10.1016/S0166-218X(99)00039-6.

54 Alasdair Urquhart. The depth of resolution proofs. *Studia Logica*, 99(1-3):349–364, 2011. doi:10.1007/s11225-011-9356-9.

55 Boris Weisfeiler. *On Construction and Identification of Graphs*, volume 558 of *Lecture Notes in Mathematics*. Springer, 1976. doi:10.1007/BFb0089374.

56 Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia, Seriya 2*, 9, 1968. Translation from Russian into English by Grigory Ryabov available under https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

Limits of CDCL Learning via Merge Resolution

Marc Vinyals
University of Auckland, New Zealand

Chunxiao Li
University of Waterloo, Canada

Noah Fleming
Memorial University of Newfoundland, St. John's, Canada

Antonina Kolokolova
Memorial University of Newfoundland, St. John's, Canada

Vijay Ganesh
University of Waterloo, Canada

Abstract

In their seminal work, Atserias et al. and independently Pipatsrisawat and Darwiche in 2009 showed that CDCL solvers can simulate resolution proofs with polynomial overhead. However, previous work does not address the tightness of the simulation, i.e., the question of how large this overhead needs to be. In this paper, we address this question by focusing on an important property of proofs generated by CDCL solvers that employ *standard learning schemes*, namely that the derivation of a learned clause has at least one inference where a literal appears in both premises (aka, a merge literal). Specifically, we show that proofs of this kind can simulate resolution proofs with at most a linear overhead, but there also exist formulas where such overhead is necessary or, more precisely, that there exist formulas with resolution proofs of linear length that require quadratic CDCL proofs.

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Constraint and logic programming

Keywords and phrases proof complexity, resolution, merge resolution, CDCL, learning scheme

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.27

Related Version *Full Version*: https://arxiv.org/abs/2304.09422

Acknowledgements The authors are grateful to Yuval Filmus and a long list of participants in the program *Satisfiability: Theory, Practice, and Beyond* at the Simons Institute for the Theory of Computing for numerous discussions. We also thank the SAT 2023 reviewers for their helpful suggestions. This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.

1 Introduction

Over the last two decades, CDCL SAT solvers have had a dramatic impact on many areas of software engineering [10], security [13, 28], and AI [7]. This is due to their ability to solve very large real-world formulas that contain upwards of millions of variables and clauses [17]. Both theorists and practitioners have expended considerable effort in understanding the CDCL algorithm and the reasons for its unreasonable effectiveness in the context of practical applications. While progress has been made, many questions remain unanswered.

Perhaps the most successful set of tools for understanding the CDCL algorithm comes from proof complexity, and a highly influential result is that shows that idealized models of CDCL are polynomially equivalent to the resolution proof system, proved independently by Atserias, Fichte, and Thurley [2], and Pipatsrisawat and Darwiche [23], building on initial results by Beame et al. [5] and Hertel et al. [15]. (See also a recent alternative proof

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 27; pp. 27:1–27:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by Beyersdorff and Böhm [6].) Such simulation results are useful because they provide us with a much simpler object (resolution proofs) to analyze than CDCL with its complicated implementation details.

Proof complexity models of CDCL need to assume a few traits of the algorithm. A series of papers have focused on understanding which of the assumptions are needed for a simulation of resolution to hold, often introducing refinements of resolution along the way. For instance, the question of whether restarts are needed, while still open, has been investigated at length, and the pool resolution [26] and RTL [9] proof systems were devised to capture proofs produced by CDCL solvers that do not restart. The importance of decision heuristics has also been explored, with results showing that neither static [21] nor VSIDS-like [27] ordering of variables are enough to simulate resolution in full generality (unless VSIDS scores are periodically erased [16]). In the case of static ordering, the (semi-)ordered resolution proof system [21] was used to reason about such variants of CDCL solvers.

But even if we stay within the idealized model, it is not clear how efficient CDCL is in simulating resolution. The analysis of Pipatsrisawat and Darwiche gives an $O(n^4)$ overhead – that is, if a formula over n variables has a resolution refutation of length L, then a CDCL proof with no more than $O(n^4 L)$ steps exists. Beyersdorff and Böhm [6] improved the overhead to $O(n^3)$, and it is unclear whether it can be further reduced. Furthermore, to the best of our knowledge, prior to our paper, we did not even know if the overhead can be avoided altogether.

1.1 Learning Schemes in CDCL and Connection with Merges

A common feature of CDCL solvers is the use of 1-empowering learning schemes [22, 2]: that is, they only learn clauses which enable unit propagations that were not possible before. An example of 1-empowering learning scheme is the popular First Unique Implication Point (1UIP) learning scheme [18]. To model this behavior we build upon a connection between 1-empowerment, and merges [1], i.e., resolution steps involving clauses with shared literals.

Nearly every CDCL solver nowadays uses the 1UIP learning scheme, where conflict analysis starts with a clause falsified by the current state of the solver and sequentially resolves it with clauses responsible for unit propagations leading to the conflict, until the clause becomes *asserting*, i.e., unit immediately upon backjumping.

Descriptions of early implementations of CDCL solvers [18, 20] already remark on the importance of learning an asserting clause, since that nudges the solver towards another part of the search space, and consequently early alternative learning schemes explored learning many kinds of asserting clauses. First observe that conflict analysis can be extended to produce other asserting clauses that appear after the 1UIP during conflict analysis such as intermediate UIPs and the last UIP [4]. The early solver GRASP can even learn multiple UIP clauses from a single conflict. While there is empirical evidence that it is often best to stop conflict analysis at the 1UIP [29], recent work has identified conditions where it is advantageous to continue past it [14] (see also the discussion of learning schemes therein).

Ryan [24, §2.5] also observed empirically that clause quality is negatively correlated with the length of the conflict analysis derivation and considered the opposite approach, that is, learning clauses that appear before the 1UIP during conflict analysis in addition to the 1UIP. This approach is claimed to be useful for some empirical benchmarks but, like any scheme that learns multiple clauses, slows down Boolean constraint propagation (BCP) in comparison to a scheme that learns just the 1UIP.

Later works provide a more theoretically oriented approach to understanding the strength of 1UIP and to learning clauses that appear before the 1UIP [12, 22]. In particular, and highly relevant for our discussion, Pipatsrisawat and Darwiche identified 1-empowerment as a fundamental property of asserting clauses. Furthermore they identified a connection between 1-empowering clauses and merges, and used the simplicity of checking for merges as an approximation for 1-empowerment.

An orthogonal approach is to extend the 1UIP derivation by resolving it with clauses other than those that would usually be used during conflict analysis [3]. A prominent example is clause minimization [25], where literals are eliminated from the 1UIP clause by resolving it with the appropriate input clauses, independently of their role in the conflict, so the resultant clause that is actually learned is a shorter and therefore stronger version of the 1UIP.

Furthermore, a relation between merges and unit-resolution completeness has also been observed in the context of knowledge compilation [11]. Finally, the amount of merges directly inferable from a formula (i.e., in a single resolution step) has been proposed, under the name of mergeability, as a measure to help explain the hardness of a formula based on both controlled experiments as well as analysis of real-world instances [30].

To summarize, merges are relevant in the context of CDCL learning schemes for the following reason: all practical CDCL learning schemes either produce a 1-empowering clause or extend one, and since 1-empowering clauses always contain a merge in its derivation, we have that all practical learning schemes produce a clause that contains a merge in its derivation, which is exactly the property imposed by the proof systems we introduce below.

1.2 Our Contributions

As mentioned earlier, we build upon a connection between 1-empowerment and merges [22, 2], and introduce a DAG-like version of Andrews' tree-like merge resolution[1] which includes CDCL with an arbitrary 1-empowering learning scheme. This is because for any 1-empowering clause, at least one step in its resolution derivation must resolve two clauses that share a common literal: a *merge* step in the sense of Andrews [1]. This is precisely the condition that our merge resolution proof system enforces. Clause minimization procedures, as long as they are applied on top of 1-empowering clauses, are also modelled by merge resolution.

We prove that, on the one hand, merge resolution is able to simulate resolution with only a linear overhead. On the other hand, we show a quadratic separation between resolution and merge resolution; that is there exist formulas with resolution proofs of linear length that require merge resolution proofs of quadratic length. The practical consequence of this pair of results is that CDCL may be polynomially worse than resolution because of the properties of a standard learning scheme, but the blow-up due to these properties is not more than linear.

We also consider weaker proof systems, all of which contain 1UIP (and do so with finer granularity), but not necessarily other asserting learning schemes. A technical point of interest is that we work with proof systems that are provably not closed under restrictions, which is unusual in proof complexity. This fact forces our proof to exploit syntactic properties of the proof system, as opposed to relying on more convenient semantic properties.

2 Preliminaries

A literal is either a variable $x^1 = x$ or its negation $x^0 = \overline{x}$. A clause is a disjunction of literals, and a CNF formula is a conjunction of clauses. The support of a clause or vars(C) is the set of variables it contains. A resolution derivation from a formula F is a sequence of clauses $\eta = C_1, \ldots, C_L$ such that C_i is either an axiom in F or it is the conclusion of applying the resolution rule

[1] Andrews referred to this simply as merge resolution, however as his system was tree-like, we feel that it is more in keeping with modern terminology to refer to it as *tree-like* merge resolution.

$$\mathrm{Res}(A \vee x, B \vee \overline{x}, x) = \mathrm{Res}(A \vee x, B \vee \overline{x}) = A \vee B$$

on two premises C_j, C_k with $j, k < i$. The unique variable x that appears with opposite signs in the premises of a resolution inference is called the pivot. If furthermore there is a *literal* common to A and B the resolvent is called a merge. If instead of being the result of a syntactic inference we allow C_i to be any clause semantically implied by C_j and C_k, even if C_j and C_k might not be resolvable, then we say η is a semantic resolution derivation. A derivation is a refutation if its last clause is the empty clause $\bot$. We denote $\eta[a, b] = \{C_i \in \eta \mid i \in [a, b]\}$.

We assume that every clause in a derivation is annotated with the premises it is obtained from, which allows us to treat the proof as a DAG where vertices are clauses and edges point from premises to conclusions. When this DAG is a tree we call a derivation tree-like, and when it is a centipede (i.e., a maximally unbalanced tree) we call it input.

A derivation is unit if in every inference at least one of the premises is a unit clause consisting of a single literal. Since neither input nor unit resolution are complete proof systems, we write $F \vdash_i C$ (respectively $F \vdash_1 C$) to indicate that there exists an input (resp. unit) resolution derivation of C from F.

A clause C syntactically depends on an axiom A with respect to a derivation η if there is a path from A to C in the DAG representation of η. This does not imply that A is required to derive C, since a different derivation might not use A.

A restriction – or more formally a variable substitution, since we allow mapping variables to other variables – is a mapping $\rho \colon X \to X \cup \{0, 1\}$, successively extended to literals, clauses, formulas, and derivations, simplifying where needed. We write $\rho(x) = *$ as a shorthand for $\rho(x) = x$. It is well-known that if η is a resolution derivation from F and ρ is a restriction, then $\eta\restriction_\rho$ is a semantic resolution derivation from $F\restriction_\rho$.

It is convenient to leave satisfied clauses in place in a derivation that is the result of applying a restriction to another derivation so that we can use the same indices to refer to both derivations. To do that we use the symbol 1 and treat it as a clause that always evaluates to true, is not supported on any set, does not depend on any clause, and cannot be syntactically resolved with any clause.

A semantic derivation can be turned into a syntactic derivation by ignoring unnecessary clauses. Formally, if η is a semantic resolution derivation, we define its syntactic equivalent $s(\eta)$ as the syntactic resolution derivation obtained by replacing each clause of $C \in \eta$ by a clause $s(C)$ as follows. If C is an axiom then $s(C) = C$. Otherwise let A and B be the parents of C. If $s(A) \vDash C$ we set $s(C) = s(A)$, analogously with $s(B)$. Otherwise we set $s(C) = \mathrm{Res}(s(A), s(B))$. It is not hard to see that for each $C_i \in \eta$, $s(C_i) \vDash C_i$.

2.1 CDCL

We need to define a few standard concepts from CDCL proofs. An in-depth treatment can be found in the Handbook of Satisfiability [8]. Let F be a CNF formula, to which we refer as a clause database. A trail τ is a sequence of tuples $(x_{j_i} = b, C_i)$ where C_i is either a clause in F or the special symbol d representing a decision. We denote by $\alpha_{<i}$ the assignment $\{x_{j_i} = b \mid i' < i\}$. A trail is valid with respect to F if for every position i that is not a decision we have that $C_i\restriction_{\alpha_{<i}} = x_{j_i}^b$, in which case we say that C_i propagates $x_{j_i}^b$, and if for every decision i there is no clause $C \in F$ such that $C\restriction_{\alpha_{<i}} = x^b$.

We denote by $\mathrm{dl}(i) = \mathrm{dl}(i-1) + [\![C_i = d]\!]$ the decision level at position i, that is the number of decisions up to i. Here $[\![C_i = d]\!]$ is the indicator of the event that the ith variable on the trail was set by a decision. We mark the position of the last decision in a trail by i^*.

A clause C is asserting if it is unit at the last decision in the trail, that is $C\restriction_{\alpha_{<i^*}} = x^b$. It is 1-empowering if C is implied by F and can lead to new unit propagations after being added to F, that is if there exists a literal $\ell \in C$ such that for some $A \in \{\bot, \ell\}$, it holds that $F \wedge \overline{C \setminus \ell} \nvdash_1 A$. If a clause is not 1-empowering then we say it is absorbed by F.

Given a clause $D_{|\tau|}$ falsified by a trail τ, the conflict derivation is an input derivation $D_{|\tau|}, D_{|\tau|-1}, \ldots, D_k$ where $D_{i-1} = \mathrm{Res}(D_i, C_i, x_{j_i})$ if $C_i \neq d$ and $x_{j_i} \in D_i$, and $D_{i-1} = D_i$ otherwise. The first (i.e., with the largest index) asserting clause in the derivation is called the 1UIP. Note that D_{i^*} is always asserting (because D_{i^*} is falsified by $\alpha_{\leq i^*}$ but not by $\alpha_{<i^*}$), therefore we can assume that the 1UIP always has index at least i^*.

We call a sequence of input derivations *input-structured* if the last clause of each derivation can be used as an axiom in successive derivations. The last clause of each but the last derivation is called a lemma. A CDCL derivation is an input-structured sequence of conflict derivations, where learned clauses are lemmas. This definition is similar to that of Resolution Trees with Input Lemmas [9], with the difference that the sequence only needs to be ordered, without imposing any further tree-structure on the global proof.

The following Lemmas highlight the practical relevance of merges by relating them to 1UIP, asserting, and 1-empowering clauses.

▶ **Lemma 1** ([22, Proposition 2]). *If a clause is asserting, then it is 1-empowering.*[2]

▶ **Lemma 2** ([2, Lemma 8]). *If $A \vee x$ and $B \vee \overline{x}$ are absorbed but $A \vee B$ is 1-empowering, then $A \vee B$ is a merge. In particular, if a clause is 1-empowering, then it contains a merge in its derivation.*

▶ **Lemma 3.** *The 1UIP clause is a merge.*

Proof. Let $D_i = \mathrm{Res}(C_{i+1}, D_{i+1})$ be the 1UIP. Every clause in F that is not already satisfied by $\alpha_{<i^*}$, and in particular C_i for $i > i^*$ and $D_{|\tau|}$, contains at least two literals at the last decision level, otherwise it would have propagated earlier. This also applies to clauses $D_{i+1}, \ldots, D_{|\tau|}$, since they are not asserting.

We accounted for 4 literals at the last decision level present in the premises of D_i, of which 2 are not present in the conclusion because they are the pivots. In order for D_i to contain only one literal at the last decision level, the remaining two literals must be equal. ◀

3 Merge Resolution

Andrews' definition of merge resolution [1] considers tree-like proofs with the additional restriction that in every inference at least one premise is an axiom or a merge. He also observes that such derivations can be made input-structured.

▶ **Observation 4** ([1]). *A tree-like merge resolution derivation can be decomposed into an input-structured sequence where all the lemmas are merges.*

This observation is key when working with such derivations, as is apparent in Section 4, to the point that we define our proof systems in terms of the *input-structured* framework. Every resolution proof can be thought of as being input-structured if we consider it as a sequence of unit-length input resolutions and every clause as a lemma; it is when we impose restrictions on which clauses are permitted as lemmas that we obtain different proof systems.

Andrews' main result is that the merge restriction does not affect tree-like resolution.

[2] The original result does not prove 1-consistency, but the proof is analogous.

▶ **Lemma 5** ([1, Lemma 5]). *If there is a tree-like resolution derivation of C of length L where at most the root is a merge, then there is an input resolution derivation of some $C' \subseteq C$ of length at most L.*

▶ **Theorem 6** ([1, Theorem 1]). *If there is a tree-like resolution derivation of C of length L, then there is a tree-like merge resolution derivation of some $C' \subseteq C$ of length at most L.*

If we lift the tree-like restriction from the input-structured view of merge resolution proofs we obtain a proof system between tree- and DAG-like resolution where clauses can be reused (i.e., have outdegree larger than 1) if and only if they are merges or, in other words, lemmas in the input-structured decomposition. As a consequence of Lemma 3 this proof system already includes CDCL refutations produced by solvers that use the 1UIP learning scheme. In order to model all asserting learning schemes we allow reusing clauses that contain a merge not only when they are inferred but anywhere in their derivation.

▶ **Definition 7.** *A merge resolution derivation is an input-structured sequence of input resolution derivations where all derivations but the last contain a merge.*

It follows from Lemmas 1 and 2 that refutations produced by solvers that use any asserting learning scheme are in merge resolution form.

We immediately have from the simulation of resolution by CDCL [23, 2] that merge resolution polynomially simulates standard resolution. In Section 4 we make this simulation more precise and prove that the simulation overhead can be made linear, and in Section 5 that the simulation is optimal because there exist formulas that have resolution refutations of linear length but require merge resolution refutations of quadratic length.

4 Simulation

As an auxiliary tool to simulate resolution in merge resolution we define the input-resolution closure of a set G, denoted $\mathrm{Cl_i}(G) = \{C \mid \exists C' \subseteq C,\, G \vdash_i C'\}$, as the set of clauses derivable from G via input resolution plus weakening. It is well-known that, since input resolution derivations can be assumed to be regular – using each variable at a pivot at most once – without loss of generality, we can also assume them to have length at most linear in the number of variables.

▶ **Observation 8.** *If G is a CNF formula over n variables and $C \in \mathrm{Cl_i}(G)$ then there is a regular input resolution derivation of some $C' \subseteq C$ from G of length at most n.*

Combining Theorem 6 with the idea that in order to simulate a resolution derivation we do not need to generate each clause, but only do enough work so that in the following steps we can pretend that we had derived it [23, 2], we can prove that merge resolution simulates resolution with at most a multiplicative linear overhead in the number of variables.

▶ **Theorem 9.** *If F is a CNF formula over n variables that has a resolution refutation of length L then it has a merge resolution refutation of length $\mathrm{O}(nL)$.*

Proof. Let $\pi = (C_1, \ldots, C_L)$ be a resolution refutation. We construct a sequence of sets $F = G_0 \subseteq \cdots \subseteq G_L$ with the following properties.

1. $G_t \setminus F$ is the set of lemmas in a merge resolution derivation from F of length at most $(2n+1)t$.
2. $\pi[1,t] \subseteq \mathrm{Cl_i}(G_t)$.

This is enough to prove the theorem: since $\bot \in \mathrm{Cl_i}(G_L)$ we can obtain $\bot$ from G_L in length n, so the total length of the refutation is $(2n+1)L+n$.

We build the sets by induction, starting with $G_0 = F$. Assume we have built G_t and let $C = C_{t+1}$. If $C \in \mathrm{Cl_i}(G_t)$ we set $G_{t+1} = G_t$ and we are done. Otherwise $C = \mathrm{Res}(A, B)$ with $A, B \in \pi[1, t]$, and by induction we have $A, B \in \mathrm{Cl_i}(G_t)$, therefore by Observation 8 there are input resolution derivations of $A' \subseteq A$ and $B' \subseteq B$ of length at most n. Since neither $A' \vDash C$ nor $B' \vDash C$, A' and B' can be resolved and therefore there is a tree-like derivation η of $C' \subseteq C$ from G_t of length at most $2n+1$. By Theorem 6 there is a tree-like merge resolution derivation η' of $C'' \subseteq C$ from G_t of length at most $2n+1$. By Observation 4 the derivation η' can be decomposed into a sequence of input derivations of total length at most $2n+1$. Let E be the lemmas in that sequence and set $G_{t+1} = G_t \cup E$. We have that $C \in \mathrm{Cl_i}(F \cup E) \subseteq \mathrm{Cl_i}(G_{t+1})$, and that we can obtain E from G_t in at most $2n+1$ steps. Thus G_{t+1} has all the required properties. ◀

5 Separation

We prove the following separation between standard and merge resolution.

▶ **Theorem 10.** *There exists a family of formulas F_n over $\mathrm{O}(n \log n)$ variables and $\mathrm{O}(n \log n)$ clauses that have resolution refutations of length $\mathrm{O}(n \log n)$ but every merge resolution refutation requires length $\Omega(n^2 \log n)$.*

5.1 Formula

Let ℓ, m, n be positive integers. We have variables x_i for $i \in [m\ell - 1]$ and $w_{j,k}$ for $j \in [\ell]$ and $k \in [n]$. For convenience we define $x_0 = 1$ and $x_{m\ell} = 0$, which are not variables. Let $X = \{x_i \mid i \in [m\ell - 1]\}$, $W_j = \{w_{j,k} \mid k \in [n]\}$ and $W = \bigcup_{j \in [\ell]} W_j$. For each $j \in [\ell]$ we build the following gadget:

$$w_{j,k} = w_{j,k+1} \qquad \text{for } k \in [n-1] \tag{1}$$

Each equality is expanded into the two clauses $B_{j,k,1} = w_{j,k} \vee \overline{w_{j,k+1}}$ and $B_{j,k,0} = \overline{w_{j,k}} \vee w_{j,k+1}$, and we collectively call them $\mathcal{W} = \{B_{j,k,b} \mid j \in [\ell], k \in [n-1], b \in \{0,1\}\}$. Observe that the j-th gadget implies $w_{j,1} = w_{j,n}$. Additionally we build the following gadget:

$$(w_{1,1} = w_{1,n}) \to x_1 \tag{2}$$

$$(w_{\hat{\imath},1} = w_{\hat{\imath},n}) \to (x_{i-1} \to x_i) \qquad \text{for } i \in [2, m\ell - 1] \tag{3}$$

$$(w_{\ell,1} = w_{\ell,n}) \to \overline{x_{m\ell-1}} \tag{4}$$

where $\hat{\imath} \in [\ell]$ denotes the canonical form of $i \pmod{\ell}$. Each constraint is expanded into the two clauses $A_{i,1} = w_{\hat{\imath},1} \vee w_{\hat{\imath},n} \vee \overline{x_{i-1}} \vee x_i$ and $A_{i,0} = \overline{w_{\hat{\imath},1}} \vee \overline{w_{\hat{\imath},n}} \vee \overline{x_{i-1}} \vee x_i$, and we collectively call them $\mathcal{X} = \{A_{i,b} \mid i \in [m\ell], b \in \{0,1\}\}$. The formula $\mathcal{X} \cup \mathcal{W}$ is called $F_{\ell,m,n}$.

5.2 Upper Bound

It is not hard to see that there is a resolution refutation of $F_{\ell,m,n}$ of length $\mathrm{O}(\ell \cdot (m+n))$. Indeed, we first derive the two clauses representing $w_{j,1} = w_{j,n}$ for each $j \in [\ell]$, which requires $\mathrm{O}(n\ell)$ steps:

$$\dfrac{\dfrac{\dfrac{\dfrac{w_{j,1} \lor \overline{w_{j,2}} \qquad w_{j,2} \lor \overline{w_{j,3}}}{w_{j,1} \lor \overline{w_{j,3}}}}{\vdots}}{w_{j,1} \lor \overline{w_{j,n-1}}} \qquad w_{j,n-1} \lor \overline{w_{j,n}}}{w_{j,1} \lor \overline{w_{j,n}}} \tag{5}$$

Then we resolve each of the $\mathcal{X}$ axioms with one of these clauses, appropriately chosen so that we obtain pairs of clauses of the form $w_{i}^{b} \lor \overline{x_{i-1}} \lor x_i$ for $i \in [m\ell]$, and resolve each pair to obtain the chain of implications $x_1, \ldots, x_i \rightarrow x_{i+1}, \ldots, \overline{x_{n\ell-1}}$ in $\mathrm{O}(m\ell)$ steps.

$$\dfrac{\dfrac{w_{\hat{\imath},1} \lor \overline{w_{\hat{\imath},n}} \qquad w_{\hat{\imath},1} \lor w_{\hat{\imath},n} \lor \overline{x_{i-1}} \lor x_i}{w_{\hat{\imath},1} \lor \overline{x_{i-1}} \lor x_i} \qquad \dfrac{\overline{w_{\hat{\imath},1}} \lor w_{\hat{\imath},n} \qquad \overline{w_{\hat{\imath},1}} \lor \overline{w_{\hat{\imath},n}} \lor \overline{x_{i-1}} \lor x_i}{\overline{w_{\hat{\imath},1}} \lor \overline{x_{i-1}} \lor x_i}}{\overline{x_{i-1}} \lor x_i} \tag{6}$$

Since we have derived a chain of implications x_1, $x_1 \rightarrow x_2$, $\ldots$, $x_{m\ell-1} \rightarrow x_{m\ell-1}$, $\overline{x_{m\ell-1}}$ we can complete the refutation in $\mathrm{O}(m\ell)$ more steps. Let us record our discussion.

▶ **Lemma 11.** *$F_{\ell,m,n}$ has a resolution refutation of length $\mathrm{O}(\ell \cdot (m+n))$.*

Before we prove the lower bound let us discuss informally what are the natural ways to refute this formula in merge resolution, so that we understand which behaviours we need to rule out.

If we try to reproduce the previous resolution refutation, since we cannot reuse the clauses representing $w_{j,1} = w_{j,n}$ because they are not merges, we have to rederive them each time we need them, which means that it takes $\mathrm{O}(mn\ell)$ steps to derive the chain of implications $x_1, \ldots, x_i \rightarrow x_{i+1}, \ldots, \overline{x_{n\ell-1}}$. We call this approach *refutation 1*. This refutation has merges (over $w_{\hat{\imath},1}$, x_{i-1}, and x_i) when we produce $w_{\hat{\imath},1}^{b} \lor \overline{x_{i-1}} \lor x_i$, and (over x_{i-1} and x_i) when we produce $\overline{x_{i-1}} \lor x_i$, but since we never reuse these clauses the refutation is in fact tree-like.

An alternative approach, which we call *refutation 2*, is to start working with the $\mathcal{X}$ axioms instead. In this proof we clump together all of the repeated constraints of the form $w_{j,1} \neq w_{j,n}$ for every $j \in [\ell]$, and then resolve them out in one go. In other words, we first derive some clausal encoding of the sequence of constraints

$$D_i = \left(\bigvee_{\hat{\imath} \in [\min(i,\ell)]} w_{\hat{\imath},1} \neq w_{\hat{\imath},n} \right) \lor x_i \qquad \text{for } i \in [m\ell] \ , \tag{7}$$

where D_i can be obtained from D_{i-1} and the pair of $\mathcal{X}$ axioms $A_{i,b}$, then resolve away the inequalities from $D_{m\ell} = \bigvee_{j \in [\ell]} w_{j,1} \neq w_{j,n}$ using the $\mathcal{W}$ axioms. However, representing any of the constraints D_i for $i \geq \ell$ requires 2^{ℓ} clauses, which is significantly larger than $mn\ell$ and even superpolynomial for large enough ℓ, so this refutation is not efficient either. Note that this refutation has merges (over W variables) each time that we derive D_i with $i \geq \ell$.

A third and somewhat contrived way to build a refutation is to derive the pair of clauses representing $w_{j,1} = w_{j,n}$ using a derivation whose last step is a merge, so that they can be reused. Each of these clauses can be derived individually in $\mathrm{O}(mn\ell)$ steps, for a total of $\mathrm{O}(mn\ell^2)$ steps, by slightly adapting refutation 1, substituting each derivation of $x_i \rightarrow x_{i+1}$ by a derivation of $w_{j,1} \lor \overline{w_{j,n}} \lor \overline{x_i} \lor x_{i+1}$ whenever $i \equiv j \pmod{\ell}$ so that at the end we obtain $w_{j,1} \lor \overline{w_{j,n}}$ instead of the empty clause. Such a substitution clause can be obtained, e.g., by resolving $w_{j,1} \lor w_{j,2} \lor \overline{x_i} \lor x_{i+1}$ with $\overline{w_{j,2}} \lor \overline{w_{j,n}} \lor \overline{x_i} \lor x_{i+1}$ as follows

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\dfrac{w_{j,2} \lor \overline{w_{j,3}} \quad w_{j,3} \lor \overline{w_{j,4}}}{w_{j,2} \lor \overline{w_{j,4}}} \\ \vdots\end{matrix}}{w_{j,2} \lor \overline{w_{j,n-1}}} \qquad w_{j,n-1} \lor \overline{w_{j,n}}}{w_{j,2} \lor \overline{w_{j,n}}} \qquad w_{i,1} \lor w_{i,n} \lor \overline{x_{i-1}} \lor x_i}{w_{i,1} \lor w_{i,2} \lor \overline{x_{i-1}} \lor x_i} \qquad \dfrac{w_{i,1} \lor \overline{w_{i,2}} \quad \overline{w_{i,1}} \lor \overline{w_{i,n}} \lor \overline{x_{i-1}} \lor x_i}{\overline{w_{i,2}} \lor \overline{w_{i,n}} \lor \overline{x_{i-1}} \lor x_i}$$

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \\ w_{i,1} \lor \overline{w_{i,n}} \lor \overline{x_{i-1}} \lor x_i \tag{8}$$

After deriving $w_{j,1} = w_{j,n}$ as merges we follow the next steps of refutation 1 and complete the refutation in $\mathrm{O}(m\ell)$ steps. We call this *refutation 3*.

Observe that the minimum length of deriving the clauses representing $w_{j,1} = w_{j,n}$ is only $\mathrm{O}(n)$, even in merge resolution, so if we only used the information that refutation 3 contains these clauses we would only be able to bound its length by $\Omega(\ell \cdot (m+n))$. Therefore when we compute the hardness of deriving a clause we need to take into account not only its semantics but how it was obtained syntactically.

5.3 Lower Bound

Before we begin proving our lower bound in earnest we make two useful observations.

▶ **Lemma 12.** *Let η be a resolution derivation that only depends on $\mathcal{W}$ axioms. Then η does not contain any merges, and all clauses are supported on W.*

Proof. We prove by induction that every clause in η is of the form $w_{j,k} \lor \overline{w_{j,k'}}$ with $k \neq k'$. This is true for the axioms. By induction hypothesis, a generic resolution step over $w_{j,k}$ is of the form

$$\dfrac{w_{j,k} \lor \overline{w_{j,k'}} \qquad \overline{w_{j,k}} \lor w_{j,k''}}{w_{j,k''} \lor \overline{w_{j,k'}}} \tag{9}$$

and in particular is not a merge. ◀

▶ **Lemma 13.** *Let η be a resolution derivation of a clause C supported on W variables that uses an $\mathcal{X}$ axiom. Then η uses at least one $A_{i,b}$ axiom for each $i \in [m\ell]$.*

Proof. We prove the contrapositive and assume that there is an axiom $A_{i,b}$ that is used, and either both $A_{i+1,0}$ and $A_{i+1,1}$ are not used, or both $A_{i-1,0}$ and $A_{i-1,1}$ are not. In the first case the literal x_i appears in every clause in the path from $A_{i,b}$ to C, contradicting that C is supported on W variables. Analogously with literal $\overline{x_{i-1}}$ in the second case. ◀

At a high level, our first step towards proving the lower bound is to rule out that refutations like refutation 2 can be small, and to do that we show that wide clauses allow for very little progress. This is a common theme in proof complexity, and the standard tool is to apply a random restriction to a short refutation in order to obtain a narrow refutation. However, merge resolution is not closed under restrictions, as we prove later in Corollary 24, and because of this we need to argue separately about which merges are preserved.

We observed that derivation fragments where no X variable appears do not contain any merges, but we cannot claim that *clauses* where no X variable appears are not merges. However, refutation 3 suggests that deriving a clause supported on W variables that depends on $\mathcal{X}$ axioms should still be hard, so we restrict our attention to the beginning of the refutation, before any such clause is derived. Within this first part it does hold that clauses where no X variable appears are not merges.

With this characterization in hand we then show that the resulting refutation needs to use all X variables, and picks up a pair of W variables each time that a new X variable appears. If we were to introduce the X variables in order, and since we ruled out wide clauses, every time that we use ℓ many X variables we also need to eliminate proportionately as many W variables before we move onto the next interval of X variables. We would expect to eliminate variables $\Omega(m\ell)$ times, and each elimination requires $\Omega(n)$ steps. Of course the refutation might not use X variables in order, but we can still break the proof into parts, each corresponding roughly to an interval of ℓ many X variables.

Let us begin implementing our plan by defining the class of restrictions that we use and which need to respect the structure of the formula. A restriction is an autarky [19] with respect to a set of clauses G if it satisfies every clause that it touches; in other words for every clause $C \in G$ either $C\restriction_\rho = 1$ or $C\restriction_\rho = C$. A restriction is k-respecting if it is an autarky with respect to $\mathcal{W}$ axioms, we have $F_{\ell,m,n}\restriction_\rho \cong F_{k,m,n}$ up to variable renaming, and every X variable is mapped to an X variable. Our definition of a narrow clause is also tailored to the formula at hand, and counts the number of different W-blocks that a clause C mentions. Formally $\mu(C) = |\{j \in [\ell] \mid \exists x_{j,k} \in \mathrm{vars(C)}\}|$.

▶ **Lemma 14.** *Let π be a resolution refutation of $F_{\ell,m,n}$ of length $L = \mathrm{o}((4/3)^{\ell/8})$. There exists an $\ell/4$-respecting restriction ρ such that every clause in $\pi\restriction_\rho$ has $\mu(C) \le \ell/8$.*

Proof. We use the probabilistic method. Consider the following probability distribution $\mathcal{J}$ over $\{0,1,*\}^\ell$: each coordinate is chosen independently with $\Pr[J_i = 0] = \Pr[J_i = 1] = 1/4$, $\Pr[J_i = *] = 1/2$. Given a random variable $J \sim \mathcal{J}$ sampled according to this distribution, we derive a random restriction ρ as follows: $\rho(w_{j,i}) = J_j$, $\rho(x_i) = *$ if $J_i = *$, and $\rho(x_i) = \rho(x_{i-1})$ otherwise (where $\rho(x_0) = 1$). We denote the set of indices not assigned by J by $J^{-1}(*)$.

Observe that $F_{\ell,m,n}\restriction_\rho \cong F_{|J^{-1}(*)|,m,n}$ up to variable renaming, and by a Chernoff bound we have $\Pr[|J^{-1}(*)| < \ell/4] \le e^{-\ell/16}$.

We also have, for every clause $C \in \pi$ with $\mu(C) > \ell/8$, that

$$\Pr[C\restriction_\rho \neq 1] \le (3/4)^{\mu(C)} \le (3/4)^{\ell/8} \quad . \tag{10}$$

Therefore, by a union bound over the length of the refutation, the probability that $|J^{-1}(*)| < \ell/4$ or that any clause has $\mu(C\restriction_\rho) > \ell/8$ is bounded away from 1 and we conclude that there exists a restriction ρ that satisfies the conclusion of the lemma. ◀

Note that $s(\pi\restriction_\rho)$ is a resolution refutation of $F_{n,\ell,m}\restriction_\rho$, but not necessarily a merge resolution refutation, therefore we lose control over which clauses may be reused[3]. Nevertheless, we can identify a fragment of $s(\pi\restriction_\rho)$ where we still have enough information.

▶ **Lemma 15.** *Let π be a merge resolution refutation of $F_{n,\ell,m}$ and ρ be the restriction from Lemma 14. There exists an integer t such that $\psi = s(\pi[1,t]\restriction_\rho)$ is a resolution derivation of a clause supported on W variables that depends on an $\mathcal{X}$ axiom and where no clause supported on W variables is reused.*

Proof. Let $C_t \in \pi$ be the first clause that depends on an $\mathcal{X}$ axiom and such that $D_t = s(C_t\restriction_\rho)$ is supported on W, which exists because $\perp$ is one such clause.

[3] Recall that $s(\pi)$ is the syntactic equivalent of π.

By definition of t, we have that every ancestor $D_k \in \psi$ of D_t that is supported on W variables corresponds to a clause C_k in π that only depends on $\mathcal{W}$ axioms, hence by Lemma 12 C_k is not a merge. By definition of merge resolution C_k is not reused, and by construction of $s(\cdot)$ neither is D_k.

It remains to prove that D_t depends on an $\mathcal{X}$ axiom. Since C_t depends on an $\mathcal{X}$ axiom, at least one of its predecessors C_p and C_q also does, say C_p. By definition of t, $D_p = s(C_p{\restriction_\rho})$ is not supported on W, and hence by Lemma 12 either D_p depends on an $\mathcal{X}$ axiom or $D_p = 1$. Analogously, if C_q also depends on an $\mathcal{X}$ axiom then so does $D_q = s(C_j{\restriction_\rho})$ (or it is 1) and we are done. Otherwise C_q is of the form $w_{j,k} \vee \overline{w_{j,k'}}$ and is either satisfied by ρ or left untouched. In both cases we have that $D_q \not\models C_t{\restriction_\rho}$ (trivially in the first case and because D_q contains the pivot while C_t does not in the second), hence D_t depends on D_p. ◀

Note that C_t may be semantically implied by the $\mathcal{W}$ axioms, and have a short derivation as in refutation 3, therefore we are forced to use syntactic arguments to argue that deriving C_t *using an $\mathcal{X}$ axiom* requires many resolution steps.

The next step is to break ψ into m (possibly intersecting) parts, each corresponding roughly to the part of ψ that uses $\mathcal{X}$ axioms with variables in an interval of length ℓ (by Lemma 13 we can assume that ψ contains axioms from every interval). To do this we use the following family of restrictions defined for $i \in [n]$:

$$\sigma_i(x_{i'}) = \begin{cases} 1 & \text{if } i' \leq i\ell \\ * & \text{if } i\ell < i' \leq (i+1)\ell \\ 0 & \text{if } (i+1)\ell < i' \end{cases} \qquad \sigma_i(w_{i',j}) = * \tag{11}$$

Let $X_i = X \cap \sigma_i^{-1}(*)$ and note that $F_{\ell,m,n}{\restriction_{\sigma_i}} \cong F_{\ell,1,n}$.

Clauses in ψ with many X variables could be tricky to classify, but intuitively it should be enough to look at the smallest positive literal and the largest negative literal, since these are the hardest to eliminate. Therefore we define $r(C)$ to be the following operation on a clause: literals over W variables are left untouched, all positive X literals but the smallest are removed, and all negative X literals but the largest are removed. Formally,

$$r\Big(\bigvee_{i\in A} x_i \vee \bigvee_{i\in B} \overline{x_i} \vee \bigvee_{(i,j)\in C} w_{i,j}^{b_{i,j}}\Big) = x_{\min A} \vee \overline{x_{\max B}} \vee \bigvee_{(i,j)\in C} w_{i,j}^{b_{i,j}} \tag{12}$$

where $x_{\min A}$ (resp. $\overline{x_{\max B}}$) is omitted if A (resp. B) is empty.

We need the following property of $r(C)$.

▶ **Lemma 16.** *If $C{\restriction_{\sigma_i}} \neq 1$ and* $\mathrm{vars(r(C))} \cap X_i = \emptyset$ *then $C{\restriction_{\sigma_i}}$ is supported over W variables.*

Proof. The hypothesis that $\mathrm{vars(r(C))} \cap X_i = \emptyset$ implies that the smallest positive X literal in C is either not larger than $i\ell$ or larger than $(i+1)\ell$, but the hypothesis that $C{\restriction_{\sigma_i}} \neq 1$ rules out the first case. Therefore all positive X literals are falsified by σ_i. Analogously the largest negative X literal is not larger than $i\ell$ and all negative X literals are also falsified. ◀

Now we are ready to formally define how to divide ψ.

▶ **Definition 17.** *The i-th part of ψ is the sequence ψ_i of all clauses $C \in \psi$ such that C is either*

1. *an $\mathcal{X}$ axiom not satisfied by σ_i; or*
2. *the conclusion of an inference with pivot in X_i; or*

3. *the conclusion of an inference with pivot in W that depends on an $\mathcal{X}$ axiom if $r(C)$ contains a variable in X_i; or*
4. *a clause that does not depend on $\mathcal{X}$ axioms if the* only *immediate successor of C is in ψ_i.*

For convenience we use the same indexing for ψ_i and ψ, so elements of ψ_i are not necessarily consecutive. Note that ψ_i needs not be a valid derivation.

This is the point in the proof where we use crucially that the original derivation is in merge resolution form: because clauses that do not depend on $\mathcal{X}$ axioms are not merges, they have only one successor and the definition is well-formed.

As an example, if ψ were refutation 1, whose only clause supported over W variables that depends on $\mathcal{X}$ axioms is the final empty clause and therefore satisfies the guarantees given by the conclusion of Lemma 15, then we would divide it as follows. Axioms $A_{1,b}, \ldots, A_{\ell,b}$ are all part of ψ_1 because of Item 1. The result of resolving axioms $A_{i,b}$ with the clauses representing $w_{i,1} = w_{i,n}$ in order to obtain the implications $x_i \to x_{i+1}$ for $i \in [1, \ell]$ are also part of ψ_1 because of Item 3. This implies that the $\mathcal{W}$ axioms and intermediate clauses used to derive $w_{i,1} = w_{i,n}$ are also part of ψ_1 because of Item 4. Derivations of x_{i+1} from x_i and $x_i \to x_{i+1}$ for $i \in [1, \ell]$ are part of ψ_1 because of Item 2. Similarly ψ_2 will contain the analogous clauses with $i \in [\ell + 1, 2\ell]$ and so on. Note that each part ψ_i contains derivations of $w_{j,1} = w_{j,n}$ for $j \in [\ell]$, but they refer to different copies of the same derivation.

Ideally we would like to argue that parts ψ_i are pairwise disjoint and of size $\Omega(n\ell)$, which would allow us to bound $|\psi| = \sum_{i\in[m]} |\psi_i| = \Omega(mn\ell)$. This is not quite true, but nevertheless clauses do not appear in too many different parts and we have the following bound.

▶ **Lemma 18.** *Let ψ and $\{\psi_i \mid i \in [m]\}$ be as discussed above. Then $2|\psi| \geq \sum_{i\in[m]} |\psi_i|$.*

Proof. Axioms may appear in at most two different ψ_i, and clauses obtained after resolving with an X pivot in only one. The only other clauses that depend on an $\mathcal{X}$ axiom and may appear in different ψ_i are obtained after resolving with a W pivot, but since $r(C)$ only contains two X variables, such clause only may appear in two different ψ_i. Finally, clauses that do not depend on an $\mathcal{X}$ axiom appear in the same ψ_i as one clause of the previous types, and therefore at most two different parts. ◀

To conclude the proof we need to argue that each ψ_i has size $\Omega(n\ell)$. The intuitive reason is that ψ_i must use one $\mathcal{X}$ axiom for each $j \in [(i\ell, (i+1)\ell]$, which introduces a pair of W variables from each W_j block, but since no clause contains more than $\ell/8$ such variables, we need to use enough $\mathcal{W}$ axioms to remove the aforementioned W variables. Formally we first need to extract a valid derivation from ψ_i as we do in the next lemma.

▶ **Definition 19.** *For each $i \in [m]$ let t_i be the smallest integer such that clause C_{t_i} depends on an $\mathcal{X}$ axiom (with respect to ψ), $C_{t_i}{\restriction_{\sigma_i}}$ is supported on W variables, and $C_{t_i} \in \psi_i$. If no such clause exists then we set $t_i = \infty$.*

▶ **Lemma 20.** *For each $i \in [m]$, t_i is finite and $s(\psi_i[1, t_i]{\restriction_{\sigma_i}})$ is a valid resolution derivation.*

Proof. We prove by induction that for all $k \leq \min(t_i, t)$, if the clause $C_k \in \psi$ depends on an $\mathcal{X}$ axiom and is not satisfied by σ_i, then there exists a clause $C_{k'} \in \psi_i$ with $k' \leq k$ that implies C_k modulo σ_i, that is $C_{k'}{\restriction_{\sigma_i}} \vDash C_k{\restriction_{\sigma_i}}$, and depends on an $\mathcal{X}$ axiom (over ψ).

When C_k is a non-satisfied $\mathcal{X}$ axiom we can simply take $C_{k'} = C_k$ by Item 1 of Definition 17. Otherwise let C_p and C_q be the premises of C_k in ψ and we consider a few cases.

Case 1: the pivot is an X variable. Then both premises depend on an $\mathcal{X}$ axiom (by Lemma 12).

Case 1.1: the pivot is an X_i variable. Then we can take $C_{k'} = C_k$ by Item 2 of Definition 17.

Case 1.2: the pivot is an $X_{i'}$ variable with $i' \neq i$. Then the pivot is assigned by σ_i and exactly one of the premises, say C_p, is non-satisfied. By the induction hypothesis we can take $C_{k'} = C_{p'}$.

Case 2: the pivot is a W variable. If either premise were satisfied, and since σ_i only assigns X variables, the satisfied literal would carry over to C_k, hence neither premise is satisfied.

Case 2.1: exactly one premise depends on an $\mathcal{X}$ axiom, say C_p. Then $C_{p'}$ is present in ψ_i, and by Item 4 of Definition 17 the other premise C_q is present in ψ_i if and only if the conclusion C_k is.

Case 2.2: both premises depend on an $\mathcal{X}$ axiom. Then both $C_{p'}$ and $C_{q'}$ are present in ψ_i.

Therefore in both subcases it is enough to prove that $C_k \in \psi_i$, since then we can take $C_{k'} = C_k$ and we have that $C_k\restriction_{\sigma_i}$ follows from a valid semantic resolution step. Assume for the sake of contradiction that $C_k \notin \psi_i$. Then for Item 3 of Definition 17 not to apply it must be that $r(C_k)$ does not contain any variable from X_i. By Lemma 16 $C_k\restriction_{\sigma_i}$ is a clause supported on W variables, which by definition of C_{t_i} implies that $k = t_i$. However, since the pivot is a W variable, $C_{p'}\restriction_{\sigma_i}$ is also supported on W variables and, together with the fact that $C_{p'}$ depends on an $\mathcal{X}$ axiom, this contradicts that C_{t_i} is the first such clause.

This finishes the induction argument and proves that $\psi_i[1, t_i]\restriction_{\sigma_i}$ is a valid semantic derivation, from where it follows that $s(\psi_i[1, t_i]\restriction_{\sigma_i})$ is a valid syntactic derivation. It also follows from the induction hypothesis that $t_i \leq t$: since C_t is left untouched by σ_i and depends on an $\mathcal{X}$ axiom, there exists a clause $C_{t'} \in \psi_i$ that depends on an $\mathcal{X}$ axiom and such that $C_{t'}\restriction_{\sigma_i} \vDash C_t\restriction_{\sigma_i} = C_t$, which is supported on W variables. ◀

Having established that each ψ_i is a valid derivation, we show that they are large in the following two lemmas.

▶ **Lemma 21.** *For each $i \in [m]$ the clause $C_{t_i}\restriction_{\sigma_i}$ depends on an $\mathcal{X}$ axiom with respect to derivation $s(\psi_i[1, t_i]\restriction_{\sigma_i})$.*

Proof. We prove by induction that for every clause $D_k \in s(\psi_i[1, t_i]\restriction_{\sigma_i})$, if C_k depends on an $\mathcal{X}$ axiom (over ψ) then so does D_k (over $s(\psi_i[1, t_i]\restriction_{\sigma_i})$). This is immediate when D_k is an axiom.

Otherwise fix C_k, $E_k = C_k\restriction_{\sigma_i}$, and $D_k = s(E_k)$, and let $E_p = C_p\restriction_{\sigma_i}$ and $E_q = C_q\restriction_{\sigma_i}$ be the premises of E_k in the semantic derivation $\psi_i[1, t_i]\restriction_{\sigma_i}$. When both C_p and C_q depend on an X axiom, then by hypothesis so do D_p and D_q and we are done because at least one of them is used to syntactically derive D_k. Otherwise one premise, say C_p, depends on an X axiom and the other premise, say C_q, does not. In that case, because σ_i only affects X variables, all the axioms used in the derivation of C_q are left untouched by σ_i, therefore we have that $D_q = E_q = C_q$, which contains the pivot used to derive C_k and therefore E_q alone does not imply E_k. In other words, the other premise E_p is semantically needed to derive E_k, and thus $D_p = s(E_p)$ is syntactically used to derive D_k. ◀

▶ **Lemma 22.** *Let η be a resolution derivation from $F_{\ell,1,n}$ of a clause C supported on W variables that depends on an $\mathcal{X}$ axiom. Then $|\eta| \geq (n-2)(\ell - \mu(C))/2$.*

Proof. By Lemma 13 we can assume that η uses at least one $A_{j,b}$ axiom for each $j \in [\ell]$.

Let $J = \{j \in [\ell] \mid \exists w_{j,k} \in \text{vars(C)}\}$ be the set of W blocks mentioned by C. We show that for each $j \in \overline{J} = [\ell] \setminus J$ at least $(n-2)/2$ axioms over variables in W_j appear in η, which makes for at least $(n-2)|\overline{J}|/2 = (n-2)(\ell - \mu(C))/2$ axioms.

Fix $j \in \overline{J}$ and assume for the sake of contradiction that less than $(n-2)/2$ axioms over variables in W_j appear in η. Then there exists $k \in [2, n-1]$ such that variable $w_{j,k}$ does not appear in η. Rename variables as follows: $w_{j,k'} \mapsto y_{k'}$ for $k' < k$, and $w_{j,k'} \mapsto \overline{y_{k'-n}}$ for $k' > k$. Then we can prove by induction, analogously to the proof of Lemma 12, that every clause derived from axiom $A_{j,b}$ is of the form $y_{k'} \vee \overline{y_{k''}} \vee D$ where D are literals supported outside W_j. Since that includes C, it contradicts our assumption that $j \notin J$. ◀

To conclude the proof of Theorem 10 we simply need to put the pieces together.

Proof of Theorem 10. We take as the formula family $F_{\ell=48\log n,n,n}$, for which a resolution refutation of length $\mathrm{O}(n \log n)$ exists by Lemma 11.

To prove a lower bound we assume that a merge resolution refutation π of length $L \leq n^3 = 2^{16\ell} = \mathrm{o}((4/3)^{8\ell})$ exists; otherwise the lower bound trivially holds. We apply the restriction given by Lemma 14 to π and we use Lemma 15 to obtain a resolution derivation ψ of a clause supported on W variables that uses an $\mathcal{X}$ axiom. We then break ψ into m parts ψ_i, each of size at least $n\ell/16$ as follows from Lemmas 20, 21, and 22. Finally by Lemma 18 we have $|\pi| \geq |\psi| \geq mn\ell/32 = \Omega(n^2 \log n)$. ◀

5.4 Structural Consequences

Theorem 10 immediately gives us two unusual structural properties of merge resolution. One is that proof length may decrease when introducing a weakening rule.

▶ **Corollary 23.** *There exists a family of formulas over* $\mathrm{O}(n \log n)$ *variables and* $\mathrm{O}(n \log n)$ *clauses that have merge resolution with weakening refutations of length* $\mathrm{O}(n \log n)$ *but every merge resolution refutation requires length* $\Omega(n^2 \log n)$*.*

Proof. Consider the formula $F_n \wedge \overline{z}$, where F_n is the formula given by Theorem 10 and z is a new variable. If we weaken every clause $C \in F_n$ to $C \vee z$ then we can derive $F \vee z \vdash z$ in $\mathrm{O}(n \log n)$ merge resolution steps because each inference is a merge. However, if we cannot do weakening, then $\overline{z}$ cannot be resolved with any clause in F_n and the lower bound of Theorem 10 applies. ◀

The second property is that merge resolution is not a *natural* proof systems in the sense of [5] because proof length may increase after a restriction.

▶ **Corollary 24.** *There exists a restriction* ρ *and a family of formulas over* $\mathrm{O}(n \log n)$ *variables and* $\mathrm{O}(n \log n)$ *clauses that have merge resolution refutations of length* $\mathrm{O}(n \log n)$ *but every merge resolution refutation of* $F_n\restriction_\rho$ *requires length* $\Omega(n^2 \log n)$*.*

Proof. Consider the formula $G_n = (F_n \vee z) \wedge \overline{z}$, where F_n is the formula given by Theorem 10, $F \vee z = \{C \vee z \mid C \in F\}$, and z is a new variable. As in the proof of Corollary 23 there is a merge resolution derivation of z of length $\mathrm{O}(n \log n)$ steps, while $G_n\restriction_\rho = F_n$. ◀

6 Further Proof Systems

In order to model CDCL with 1UIP more closely and possibly obtain a stronger separation we look at restricted versions of merge resolution, which in this section we refer to as Resolution with Merge Ancestors (RMA) to disambiguate it from the rest. The diagram in Figure 1 can help keeping track of these.

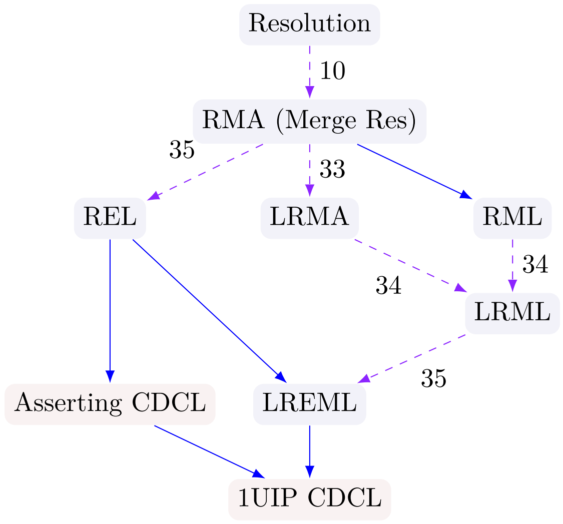


Figure 1 Relations between proof systems. A solid arrow $A \longrightarrow B$ indicates that A simulates B with no overhead. A dashed arrow $A \dashrightarrow B$ indicates that A simulates B with no overhead, but B requires linear overhead to simulate A. Statements proving separations are referenced.

▶ **Definition 25** (Definition 7, restated). *A Resolution with Merge Ancestors (RMA) derivation is an input-structured sequence of input resolution derivations where all derivations but the last contain a merge.*

Note that by Lemma 5 it does not matter if we require the sequence of derivations of an RMA derivation to be input derivations or if we allow general trees. In fact, our lower bound results hold for a more general proof system where we only ask that every clause with outdegree larger than 1 has an ancestor that is a merge. Such a proof system does not have a simple input structure, but can rather be thought of as a sequence of tree-like resolution derivations whose roots are merges, followed by a standard resolution derivation using the roots of the previous derivations as axioms.

To make the connection back to CDCL, we can define a proof system called Resolution with Empowering Lemmas that captures CDCL refutations produced by solvers that use any asserting learning scheme or 1-empowering learning scheme.

▶ **Definition 26.** *Let $C_1, \ldots, C_{L-1}$ be the lemmas of an input-structured sequence of input derivations. The sequence is a Resolution with Empowering Lemmas (REL) derivation of a formula F if C_i is 1-empowering with respect to $F \cup \{C_j : j < i\}$ for all $i \in [1, L-1]$.*

It follows from Lemma 2 that such refutations are in RMA form.

▶ **Observation 27.** *A REL derivation is a RMA derivation.*

It might seem more natural to work with the REL proof system rather than its merge-based counterparts, since REL is defined exactly through the 1-empowering property. However, while the merge property is easy to check because it is local to the derivation at hand, we can only determine if a clause is 1-empowering by looking at the full history of the derivation, in particular what the previous lemmas are. This makes REL too cumbersome to analyse. Furthermore, refutations produced by CDCL applying a clause minimization scheme on top of an asserting clause might not be in REL form, but they are still in RMA form.

We also discussed in Section 3 we that 1UIP CDCL solvers produce derivations where lemmas themselves are merges. We call this proof system Resolution with Merge Lemmas, or RML for short.

▶ **Definition 28.** *A Resolution with Merge Lemmas (RML) derivation is an input-structured sequence of input resolution derivations where all lemmas are merges.*

We can be even more restrictive and observe that input derivations produced by a CDCL solver that we describe next is that once a variable is resolved, it does not appear later in the derivation.

▶ **Definition 29.** *A resolution derivation η is strongly regular if for every resolution step i, the pivot variable x_i is not amongst the variables of any clause $C_i \in \eta[i, L]$. A sequence of derivations is locally regular if every derivation in the sequence is strongly regular. A LRML derivation (resp. LRMA) is a locally regular RML derivation (resp. RMA).*

Finally we can consider derivations that have empowering, merge lemmas and are locally regular. These still include 1UIP proofs.

▶ **Definition 30.** *A LREML derivation is a derivation that is both LRML and REL.*

All of the proof systems we defined are quadratically separated from resolution simply because they are weaker than RMA. At the same time, all of these proof systems still simulate standard resolution up to linear overhead, as we show next.

Going back to the proof of Theorem 9, we first observe that the resulting RMA refutation is in fact an RML refutation.

Recall that the proof idea is to maintain a set of clauses G_t such that all clauses in the proof up to time t can be derived from G_t by input derivation. Then, given a new clause C_{t+1} that can be obtained from G_t with a derivation η, we transform η into a merge resolution derivation using Theorem 6, and we add its lemmas to G_{t+1}. Since Theorem 6 produces RML derivations, so is the final derivation we construct.

To make the simulation work also for LREML we need the following lemma.

▶ **Lemma 31** ([23]). *If F absorbs $A \lor x$ and $B \lor \overline{x}$, then $F \vdash_i C' \subseteq A \lor B$.*

The simulation itself follows the general structure of Theorem 9, except that we need some additional work to construct G_{t+1} from G_t.

▶ **Theorem 32.** *If F is a CNF formula over n variables that has a resolution refutation of length L then it has a LREML refutation of length $\mathrm{O}(nL)$.*

Proof. Let $\pi = (C_1, \dots, C_L)$ be a resolution refutation. As we already showed it is enough to construct a sequence of sets $F = G_0 \subseteq \cdots \subseteq G_L$ such that $G_t \setminus F$ is the set of lemmas in a LREML derivation from F of length at most $(2n+1)t$, and $\pi[1,t] \subseteq \mathrm{Cl_i}(G_t)$. Assume we have built G_t and let $C = C_{t+1}$. If $C \in \mathrm{Cl_i}(G_t)$ we set $G_{t+1} = G_t$ and we are done. Otherwise we showed that there are input resolution derivations of $A' \subseteq A$ and $B' \subseteq B$ from G_t of length at most n, which we can assume are strongly regular, and that A' and B' can be resolved together.

At this point we deviate from the proof of Theorem 9. We inductively build an intermediate sequence of sets $G_t = G_t^0 \subseteq \ldots \subseteq G_t^k$ with the following properties.

1. G_t^j is the set of lemmas in a LREML derivation from G_t of length at most p.
2. A' and B' can be derived from G_t^j in at most $2n - p$ resolution steps.

The base case $G_t^0 = G_t$ is trivial. For the inductive case, let us first assume that either A' or B' is 1-empowering, say A'. Let E be the first 1-empowering clause in the derivation of A'. By Lemma 2 E is a merge, therefore we are allowed to take $G_t^{j+1} = G_t^j \cup \{E\}$. Furthermore, if A' and E could be derived from G_t^j in r and s steps, then A' can be derived from G_t^{j+1} in $r - s$ steps, simply by omitting the first s steps in the derivation.

Otherwise, by Lemma 31 we reached a set G_t^j such that $C \in \mathrm{Cl_i}(G_t^j)$. In this case we choose $k = j$ and $G_{t+1} = G_t^k$. Since G_{t+1} can be obtained in $p \leq 2n$ steps from G_t, it satisfies the required properties. This concludes both the inner and outer inductions. ◀

One consequence of the proof systems we introduce being polynomially equivalent to resolution is that they are conjectured to be incomparable with respect to the related RTL and pool resolution proof systems, since these are conjectured to be exponentially weaker than resolution. This would not be too unexpected given the different purposes of the proof systems: RTL and pool resolution were introduced to study restarts, and include proofs produced by CDCL without restarts but any kind of learning, while the purpose of merge-based proof systems is to study learning, and include proofs produced by CDCL with our without restarts but only asserting learning.

We can separate the different proof systems that we introduced using a few variations of $F_{\ell,m,n}$ where we add a constant number of redundant clauses for each $i \in [\ell]$. We present the results that we obtain next, and defer the proofs to the full version.

▶ **Proposition 33.** *There exists a family of formulas over* $\mathrm{O}(n \log n)$ *variables and* $\mathrm{O}(n \log n)$ *clauses that have RMA refutations of length* $\mathrm{O}(n \log n)$ *but every LRMA refutation requires length* $\Omega(n^2 \log n)$*.*

▶ **Proposition 34.** *There exists a family of formulas over* $\mathrm{O}(n \log n)$ *variables and* $\mathrm{O}(n \log n)$ *clauses that have RML and LRMA and refutations of length* $\mathrm{O}(n \log n)$ *but every LRML refutation requires length* $\Omega(n^2 \log n)$*.*

▶ **Proposition 35.** *There exists a family of formulas over* $\mathrm{O}(n \log n)$ *variables and* $\mathrm{O}(n \log n)$ *clauses that have LRML refutations of length* $\mathrm{O}(n \log n)$ *but every REL refutation requires length* $\Omega(n^2 \log n)$*.*

7 Concluding Remarks

In this paper, we address the question of the tightness of simulation of resolution proofs by CDCL solvers. Specifically, we show that RMA, among other flavours of DAG-like merge resolution, simulates standard resolution with at most a linear multiplicative overhead. However, contrary to what we see in the tree-like case, this overhead is necessary. While the proof systems we introduce help us explain one source of overhead in the simulation of resolution by CDCL, it is not clear if they capture it exactly. In other words, an interesting future direction would be to explore whether it is possible for CDCL to simulate some flavour of merge resolution with less overhead than what is required to simulate standard resolution.

References

1 Peter B. Andrews. Resolution with merging. *J. ACM*, 15(3):367–381, 1968.

2 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *Journal of Artificial Intelligence Research*, 40:353–373, January 2011. Preliminary version in *SAT '09*.

3 Gilles Audemard, Lucas Bordeaux, Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. A generalized framework for conflict analysis. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing – SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 21–27. Springer, 2008. doi:10.1007/978-3-540-79719-7_3.

4 Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

5 Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, December 2004. Preliminary version in *IJCAI '03*.

6 Olaf Beyersdorff and Benjamin Böhm. Understanding the relative strength of QBF CDCL solvers and QBF resolution. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 12:1–12:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.ITCS.2021.12`.

7 Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1-2):281–300, 1997.

8 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 233–350. IOS Press, 2nd edition, 2021. `doi:10.3233/FAIA200990`.

9 Samuel R. Buss, Jan Hoffmann, and Jan Johannsen. Resolution trees with lemmas: Resolution refinements that characterize DLL-algorithms with clause learning. *Logical Methods in Computer Science*, 4(4:13), December 2008.

10 Cristian Cadar, Vijay Ganesh, Peter M Pawlowski, David L Dill, and Dawson R Engler. EXE: Automatically Generating Inputs of Death. *ACM Transactions on Information and System Security (TISSEC)*, 12(2):1–38, 2008.

11 Alvaro del Val. Tractable databases: How to make propositional unit resolution complete through compilation. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94). Bonn, Germany, May 24-27, 1994*, pages 551–561. Morgan Kaufmann, 1994.

12 Nachum Dershowitz, Ziyad Hanna, and Alexander Nadel. Towards a better understanding of the functionality of a conflict-driven SAT solver. In João Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing – SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*, volume 4501 of *Lecture Notes in Computer Science*, pages 287–293. Springer, 2007. `doi:10.1007/978-3-540-72788-0_27`.

13 Julian Dolby, Mandana Vaziri, and Frank Tip. Finding Bugs Efficiently With a SAT Solver. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 195–204, 2007. `doi:10.1145/1287624.1287653`.

14 Nick Feng and Fahiem Bacchus. Clause size reduction with all-uip learning. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing – SAT 2020 – 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 28–45. Springer, 2020. `doi:10.1007/978-3-030-51825-7_3`.

15 Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively P-simulate general propositional resolution. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08)*, pages 283–290, July 2008.

16 Chunxiao Li, Noah Fleming, Marc Vinyals, Toniann Pitassi, and Vijay Ganesh. Towards a complexity-theoretic understanding of restarts in sat solvers. In *Theory and Applications of Satisfiability Testing–SAT 2020: 23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings 23*, pages 233–249. Springer, 2020.

17 João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability – Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2021. `doi:10.3233/FAIA200987`.

18 João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

19 Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discret. Appl. Math.*, 10(3):287–295, 1985. doi:10.1016/0166-218X(85)90050-2.

20 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

21 Nathan Mull, Shuo Pang, and Alexander A. Razborov. On CDCL-based proof systems with the ordered decision strategy. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT '20)*, volume 12178 of *Lecture Notes in Computer Science*, pages 149–165. Springer, July 2020.

22 Knot Pipatsrisawat and Adnan Darwiche. A new clause learning scheme for efficient unsatisfiability proofs. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1481–1484. AAAI Press, 2008. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-243.php.

23 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, February 2011. Preliminary version in *CP '09*.

24 Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004.

25 Niklas Sörensson and Armin Biere. Minimizing learned clauses. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT '09)*, volume 5584 of *Lecture Notes in Computer Science*, pages 237–243. Springer, July 2009.

26 Allen Van Gelder. Pool resolution and its relation to regular resolution and DPLL with clause learning. In *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR '05)*, volume 3835 of *Lecture Notes in Computer Science*, pages 580–594. Springer, 2005.

27 Marc Vinyals. Hard examples for common variable decision heuristics. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1652–1659, February 2020.

28 Yichen Xie and Alexander Aiken. Saturn: A SAT-Based Tool for Bug Detection. In *Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005*, pages 139–143, 2005. doi:10.1007/11513988_13.

29 Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in Boolean satisfiability solver. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '01)*, pages 279–285, November 2001.

30 Edward Zulkoski, Ruben Martins, Christoph M. Wintersteiger, Jia Hui Liang, Krzysztof Czarnecki, and Vijay Ganesh. The effect of structural measures and merges on SAT solver performance. In John N. Hooker, editor, *Principles and Practice of Constraint Programming – 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 436–452. Springer, 2018. doi:10.1007/978-3-319-98334-9_29.

Explaining SAT Solving Using Causal Reasoning

Jiong Yang ✉
National University of Singapore, Singapore

Arijit Shaw ✉
Chennai Mathematical Institute, India
IAI, TCG-CREST, Kolkata, India

Teodora Baluta ✉
National University of Singapore, Singapore

Mate Soos
National University of Singapore, Singapore

Kuldeep S. Meel
National University of Singapore, Singapore

Abstract

The past three decades have witnessed notable success in designing efficient SAT solvers, with modern solvers capable of solving industrial benchmarks containing millions of variables in just a few seconds. The success of modern SAT solvers owes to the widely-used CDCL algorithm, which lacks comprehensive theoretical investigation. Furthermore, it has been observed that CDCL solvers still struggle to deal with specific classes of benchmarks comprising only hundreds of variables, which contrasts with their widespread use in real-world applications. Consequently, there is an urgent need to uncover the inner workings of these seemingly *weak* yet *powerful* black boxes.

In this paper, we present a first step towards this goal by introducing an approach called CausalSAT, which employs causal reasoning to gain insights into the functioning of modern SAT solvers. CausalSAT initially generates observational data from the execution of SAT solvers and learns a structured graph representing the causal relationships between the components of a SAT solver. Subsequently, given a query such as whether a clause with low literals blocks distance (LBD) has a higher clause utility, CausalSAT calculates the causal effect of LBD on clause utility and provides an answer to the question. We use CausalSAT to quantitatively verify hypotheses previously regarded as "rules of thumb" or empirical findings, such as the query above or the notion that clauses with high LBD experience a rapid drop in utility over time. Moreover, CausalSAT can address previously unexplored questions, like which branching heuristic leads to greater clause utility in order to study the relationship between branching and clause management. Experimental evaluations using practical benchmarks demonstrate that CausalSAT effectively fits the data, verifies four "rules of thumb", and provides answers to three questions closely related to implementing modern solvers.

2012 ACM Subject Classification Theory of computation; Computing methodologies → Artificial intelligence

Keywords and phrases Satisfiability, Causality, SAT solver, Clause management

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.28

Related Version *Full Version*: `https://arxiv.org/abs/2306.06294`

Supplementary Material *Software (Source Code)*: `https://github.com/meelgroup/causalsat`

Funding This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004], Ministry of Education Singapore Tier 2 Grant [MOE-T2EP20121-0011], and Ministry of Education Singapore Tier 1 Grant [R-252-000-B59-114]. Teodora Baluta is also supported by the Google PhD Fellowship.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 28; pp. 28:1–28:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We are thankful to Tim van Bremen for providing detailed feedback on the early drafts of the paper and grateful to the anonymous reviewers for their constructive comments to improve this paper. We extend sincere gratitude to Jakob Nordström and other participants of the program *Satisfiability: Theory, Practice, and Beyond* at the Simons Institute for the Theory of Computing for insightful discussions. Part of the work was done during Arijit Shaw's internship at the National University of Singapore. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore, https://www.nscc.sg.

1 Introduction

Boolean Satisfiability (SAT) is a fundamental problem in computer science that involves determining whether there exists an assignment σ that satisfies a given Boolean formula F. The applications of SAT are vast and varied, including but not limited to bioinformatics [29], AI planning [22], and hardware and system verification [7,11]. The seminal work of Cook [13] demonstrated that SAT is NP-complete. Unsurprisingly, early algorithmic methods such as local search and the DPLL paradigm [14] faced significant scalability challenges in practice. In the early '90s, the introduction of Conflict Driven Clause Learning (CDCL) [45] ushered in a new era of interest from both theoreticians and practitioners. The outcome of this development is the emergence of effective heuristics that empower SAT solvers to handle challenging instances across various domains. This remarkable advancement is widely recognized as the *SAT revolution* [3,6,15,25,30–32,45].

The modern CDCL SAT solvers owe their performance to well-designed and tightly integrated core components: *branching* [25,30], *phase selection* [38], *clause learning* [4,28], *restarts* [3,18,19,24], and *learnt clause cleaning* [5,34]. Consequently, the progress has been driven largely by the continuous improvement of heuristics for these core components. The annual SAT competition [21] has provided evidence of a pattern in which the development of heuristics for one core component necessitates and encourages the design of new heuristics for other components to ensure tight integration. Such progress, however, necessitated the solvers to be complex, whereas a typical modern SAT solver is made up of approximately $15-30$ thousand lines of code and employs a wide variety of strategies to solve different types of problems. Therefore, to make further progress in the design of SAT solvers, it is crucial to understand the inner workings of these SAT solvers.

Traditional complexity-theoretic studies focus on analyzing the limitations of SAT solvers: such as determining the explanations for why SAT solvers struggle for certain instances [16]. These approaches, however, fail to inform reasons for their success in solving industrial-size instances. Several prior studies provide intuitive hypotheses on why certain SAT-solving heuristics work well on some benchmarks [34,48,53]. These hypotheses are mostly derived from the researcher or developer's intuition and empirically tested in an ad-hoc manner. The tight coupling of heuristics has been a crucial part of the development of modern solvers [47,49]. For instance, Biere et al. [9] found that certain branching heuristics work particularly well in combination with specific restart heuristics. Similarly, Oh [33] observed that in order to achieve the best solver performance, the decay factor used in activity calculations must be adjusted according to the active restart heuristics.

Motivated by the role played by empirical studies in the development of new heuristics, we take the approach of usage of data-driven methods to develop understandings of the behavior of SAT solvers. Our work relies on the CrystalBall framework [48] that provides white-box access to the execution of a state-of-the-art SAT solver, thereby enabling large-scale data collection. Given the recent development in the field of causal reasoning [35], we investigate *whether it is possible to develop a framework that may use white-box access to the execution of SAT solving to generate causal reasoning to explain the interplay among the features and heuristics of SAT solving?*

Several studies have investigated the behavior of modern solvers. Elffers et al. [16] explored the effectiveness of various solver components by employing a large set of theoretical benchmarks, with the running time serving as the parameter to evaluate their utility. Simon [46] focused on analyzing the generated clauses' utility by examining the proofs produced by a SAT solver. Kokkala et al. [23] utilized DRAT-trim-based proofs to investigate the impact of restart and learned clause cleaning heuristics on the length of trimmed proofs. They also investigated the relationship between different parameters and the probability of a clause being included in the trimmed proof. Soos et al. [48] employed CrystalBall to deduce the statistical correlation between the utility of learned clauses and SAT-solving features. Nevertheless, all of these studies were based on correlation analysis and lacked a causal reasoning perspective in understanding solver behavior.

In this work, we propose to use *causal reasoning* to answer what factors influence the utility of clauses in memory management in SAT solvers. Our approach combines prior knowledge and experimental data to derive a causal model, a graph whose nodes represent different components and heuristics of the solver, and its edges represent causal relations between them. The derived causal model explicitly encodes assumptions about the underlying phenomenon in clause memory management. The causal model allows us to query it by applying a set of principled rules known as do-calculus to compute the effect of a particular factor on the utility of clauses. These rules ensure that each query computes the right statistical quantity given the model without introducing bias in the estimates. We find that our derived models fit the data reasonably well even when less data is available. Using our framework, we were able to confirm several well-known hypotheses about solvers, which highlights its potential and assures its correctness. Additionally, we have used the framework to inquire about some unresolved questions related to the solving process, and it has provided us with answers to those inquiries. This highlights that our framework provides a fresh perspective on how to approach and analyze solvers.

We demonstrate the application of our approach to study four hypotheses that stem from observations or assumptions in prior work and encode these as causal queries. We find that clauses with lower Literal Block Distance (LBD) have greater utility; thus, lower LBD clauses should be prioritized in clause memory management. We find that small clauses have greater utility, regardless of their LBD. We also justify the hierarchical clause memory management that high LBD clauses should be cached for a short time since their utility experiences a rapid drop over time. Moreover, we verify that LBD has a larger causal impact than the size on clause utility, and hence LBD is a preferable choice for prioritizing clauses in memory management. We additionally study three novel questions that have not been formulated before, such as which branching heuristic results in a greater clause utility, to study the relationship between branching and clause management. We find that as a branching heuristic, Maple [24] leads to a greater clause utility than VSIDS [31]. To summarize, we demonstrate that causal reasoning allows one to evaluate the crucial choices in the design of SAT solvers.

2 Background and Motivation

2.1 CDCL Solvers

CDCL-based SAT solvers begin with an initially empty set of assignments, maintaining a partial assignment at each step. The solver incrementally assigns a subset of variables until the current partial assignment is unable to satisfy the current formula. At this point, the solver employs a backtracking mechanism to trace the cause of unsatisfiability, which is expressed as a conflict clause. Modern solvers frequently perform restarts, resetting the partial assignment to empty.

Modern SAT solvers may run for millions of conflicts during a single execution, with each conflict resulting in the learning of one or more clauses that are subsequently added to a database. As this database grows over time, it becomes necessary to periodically remove some of the less useful clauses to maintain solver efficiency. To accomplish this, various heuristics have been proposed that predict the usefulness of learned clauses. These heuristics, collectively known as *learned clause cleaning* heuristics, enable the solver to identify and remove unnecessary clauses, improving performance and scalability. A *branching heuristic* is employed to determine which variable to select for each decision point in the search process. A *restart heuristic* is used to determine when the solver should restart the search process from the beginning.

2.2 Learnt Clause Cleaning

One of the most crucial parts of a CDCL-based SAT solver is the learning of clauses, which occurs rapidly. In fact, a typical SAT solver can learn millions of clauses within a minute of operation. However, the sheer volume of these clauses can hinder the solver's efficiency. To address this issue, *learnt clause cleaning* heuristics were developed. These heuristics evaluate the "utility" of a clause based on some parameters and determine whether to retain or discard it. A heuristic may prioritize keeping clauses with higher utility while discarding those with lower utility to optimize the solver's performance. Previous literature proposed various parameters to determine the usefulness of a clause. Een et al. [15] employed the notion of *activity (or VSIDS)* as a surrogate for utility. Activity is a parameter that indicates a clause's involvement in recent conflicts. On the other hand, Audemard et al. [3] utilized *literal block distance (LBD)* as a proxy for utility, where LBD represents the number of unique decision levels in the clause. Despite its success, it is difficult to tell *why* LBD is a good indicator of utility and disentangle its impact from other solving heuristics. To illustrate this, we highlight some of the existing hypotheses along with novel questions about the utility of clauses below.

- Clauses with small LBD have greater utility.
- Small clauses have greater utility.
- High-LBD clause experiences a rapid drop in clause utility over time.
- LBD has a greater impact on clause utility than clause size.
- What factor, other than LBD, size, and activity, has the greatest impact on clause utility?
- Which branching heuristic results in the greatest clause utility?
- Which restart heuristic results in the greatest clause utility?

It is thus natural to ask if these hypotheses and questions explain the underlying SAT-solving behavior well. Do they provide new insights into other aspects of solving? We show that in order to correctly answer such questions, we require a principled framework to even compute the right statistical quantities from data.

2.3 Causal Model

A causal model represents the set of features that cause an outcome and the function that measures the effect of the features on the outcome. Prior work proposes prior domain knowledge such as causal invariant transformations [51], score-based learning algorithms [42], interventional causal representation learning [1, 10, 39] or learning invariant relationships from training datasets from different distributions [2, 37] to construct causal models.

As with probabilistic graphical models, the underlying representation of the causal model is (1) a directed acyclic graph (DAG) $G = (V, E)$ where the vertices V are the set of random variables $V = \{X_1, \ldots, X_n\}$ and the edges $(X_i, X_j) \in E$ represent causal relationships X_i causes X_j and (2) set of parameters Θ quantifying the relationships between the variables in V. The set of variables V can take either discrete or continuous values. We say that a subset of variables $S \subset V$ cause X_i, if $X_i = g(\{X_j\}_{j \in S}, N_Y)$ for some function g and N_Y a random variable independent of all other variables. The set of functions is parameterized by $\theta \in \Theta$. Henceforth, we will use causal models and causal graphs interchangeably. The absence of an edge from X to Y represents (conditional) independence. Due to this, causal graphs are also known as causal Bayesian networks. Causal graphs still have the probabilistic interpretation of their associative (Bayesian) counterpart: they represent a joint probability distribution of the variables in V, determined by their conditional independence relationships, i.e., $\mathcal{P}_\theta(V) = \Pi_i \Pr[X_i | pa_{X_i}], X_i \in V$, where pa_{X_i} represents the parents of the X_i vertex.

Causal Inference

The causal graph has a quantitative interpretation: we use it to answer *do-queries* that estimate the causal effect between a *treatment* variable and an *outcome* variable.

***do*-query.** We say there is a causal effect between a treatment variable X_i and an outcome variable Y if under an intervention on the treatment variable X_i, the outcome variable changes. Given a set of variables $V = \{X_1, \ldots, X_n, Y\}$, an intervention on a variable X_i is an experiment where the experimenter controls the variable $X_i \in S$ to take a value u of another independent (from other variables $\in V$) variable, i.e., $X_i = u$. More generally, we can intervene on a subset of variables $S \subset V$. This operation has been formalized as the *do* operator by Pearl [35]. The *do* operator thus changes the value of X_i while keeping every other variable in V the same, except for those directly or indirectly affected by X_i. This is akin to removing the edges of the nodes in S to their parents in the graph G, resulting in a manipulated graph. The intervention effect on the outcome variable, e.g., $\Pr[Y | do(X_i = u)]$, is then measured by data generated under the distribution of the manipulated graph. *Do*-queries are thus fundamentally different from observational or conditional queries in that it requires being able to set the value of the variable X_i to a potentially unobserved value and keeping every other unaffected variable in V the same. The framework to reason about *do*-queries has been formalized in a set of rules known as the *do*-calculus which has been shown to be sound and complete [20].

Average Treatment Effect (ATE). Given a causal graph, a treatment variable X, and an outcome variable Y, the average treatment effect measures the change of the expected value of the outcome variable when we intervene on the treatment variable and change it from a constant value b to a.

▶ **Definition 1** (Average Treatment Effect). *The average treatment effect of a variable X (called the treatment) on the target variable Y (called the outcome) is:*

$$ATE(X, Y, a, b) = \mathbb{E}\left[Y | do(X = a)\right] - \mathbb{E}\left[Y | do(X = b)\right],$$

where a, b are constants for which X is defined. We omit the constants when the query is over the domain of X. Following Definition 1, the conditional average treatment effect (CATE) measures the average outcome conditioned on a variable W of interest.

▶ **Definition 2** (Conditional Average Treatment Effect). *The conditional average treatment effect of X on Y conditioned on a variable of interest W is:*

$$CATE(X, Y, W, a, b) = \mathbb{E}\left[Y|do(X=a), W\right] - \mathbb{E}\left[Y|do(X=b), W\right].$$

Why Causal Analysis?

Deriving the right quantities to compute is challenging without a proper framework. As an example, we first consider LBD, $\mathsf{Propagation}$, i.e., the number of times a clause is involved in propagation, and $\mathsf{LastTouch}$, i.e., the number of conflicts since a clause is involved in conflict analysis. We collect data corresponding to these variables during the SAT-solving process of 80 instances by varying several of the existing heuristics. Suppose we want to compute the impact of $\mathsf{Propagation}$ on $\mathsf{LastTouch}$, that is, the expected change of $\mathsf{LastTouch}$ if $\mathsf{Propagation}$ increases by one. This quantity can be written as the expected value of $\mathsf{LastTouch}$ when $\mathsf{Propagation}$ increased from one to two: $\mathbb{E}\left[\mathsf{LastTouch} \mid \mathsf{Propagation} = 2\right] - \mathbb{E}\left[\mathsf{LastTouch} \mid \mathsf{Propagation} = 1\right] = -431.29$.

However, there is a common cause, LBD, that affects both $\mathsf{Propagation}$ and $\mathsf{LastTouch}$. Having this knowledge at hand, we need to "control" for LBD. "Controlling" for a common cause in statistical measurements means binning over the observed values of the common cause. Thus, the right quantity to compute is: $\sum_z(\mathbb{E}\left[\mathsf{LastTouch} \mid \mathsf{Propagation} = 2, \mathsf{LBD} = z\right] - \mathbb{E}\left[\mathsf{LastTouch} \mid \mathsf{Propagation} = 1, \mathsf{LBD} = z\right]) \Pr[\mathsf{LBD} = z]$. Notice that we can represent such relationships between variables using a *causal model*, i.e., a graph where the nodes are variables and the edges represent causal relations. We show the graph corresponding to our example in Figure 1a. Estimating this statistical quantity using a linear regression model that fits the data yields -2.92. Thus, our first attempt at estimating the effect did not model the underlying process precisely.

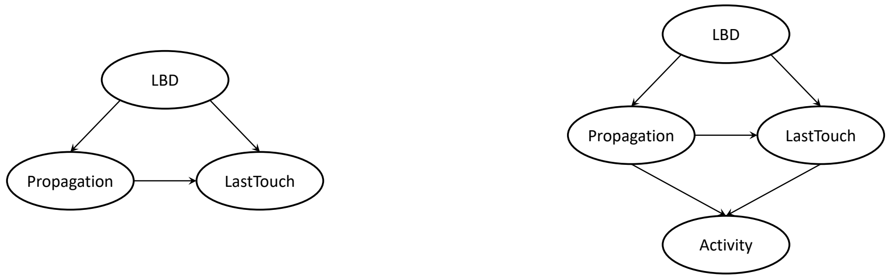
Let us now consider a fourth variable, $\mathsf{Activity}$ that is also correlated with $\mathsf{Propagation}$ and $\mathsf{LastTouch}$ (see Figure 1b). Should we control for this variable as well? Suppose we controlled for it and estimated again the effect of $\mathsf{Propagation}$ on $\mathsf{LastTouch}$ as the following:

$$\sum_{z,a}(\mathbb{E}\left[\mathsf{LastTouch} \mid \mathsf{Propagation} = 2, \mathsf{LBD} = z, \mathsf{Activity} = a\right] - \\ \mathbb{E}\left[\mathsf{LastTouch} \mid \mathsf{Propagation} = 1, \mathsf{LBD} = z, \mathsf{Activity} = a\right]) \Pr[\mathsf{LBD} = z] \Pr[\mathsf{Activity} = a]$$

On the collected data, this results in an estimated effect of -1.96. However, $\mathsf{Activity}$ should not be controlled for since it is not a common cause, but rather it is affected by both $\mathsf{Propagation}$ and $\mathsf{LastTouch}$. Intuitively, during SAT solving, the $\mathsf{Activity}$ is computed *after* these two variables. The correct estimate is $\sum_z(\mathbb{E}\left[\mathsf{LastTouch} \mid \mathsf{Propagation} = 2, \mathsf{LBD} = z\right] - \mathbb{E}\left[\mathsf{LastTouch} \mid \mathsf{Propagation} = 1, \mathsf{LBD} = z\right]) \Pr[\mathsf{LBD} = z]$ = -2.92. Over-controlling biased the estimated effect. It is thus crucial to identify the correct set of variables to control.

3 Causal Reasoning for SAT Solving

LBD, size, and activity have long been employed to predict the utility of learned clauses in SAT solvers. For instance, Kissat [8] consistently retains clauses with LBD ≤ 2, while CryptoMiniSat [49] always preserves clauses with LBD ≤ 3. These heuristics constitute a critical component and substantially enhance the performance of modern SAT solvers. Despite being considered a "rule of thumb" for solver design, whether a low-LBD clause yields higher utility than a high-LBD clause still remains uncertain.



$\sum_z(\mathbb{E}[\textsf{LastTouch}|\textsf{Propagation}{=}2,\textsf{LBD}{=}z]$
$\quad - \ \mathbb{E}[\textsf{LastTouch}|\textsf{Propagation}{=}1,\textsf{LBD}{=}z])\Pr[z]$

$\sum_{z,a}(\mathbb{E}[\textsf{LastTouch}|\textsf{Propagation}{=}2,\textsf{LBD}{=}z,\textsf{Activity}{=}a]$
$\quad - \ \mathbb{E}[\textsf{LastTouch}|\textsf{Propagation}{=}1,\textsf{LBD}{=}z,\textsf{Activity}{=}a])\Pr[z]\Pr[a]$

(a) **(b)**

Figure 1 Causal models help correctly identify the set of variables to control for. (a) Estimated effect of Propagation on LastTouch when LBD is a cause of both. (b) Over-controlling while estimating the effect of Propagation on LastTouch when Activity is introduced. We use the notation $\Pr[z]$ and $\Pr[a]$ as a shorthand for $\Pr[\textsf{LBD} = z]$ and $\Pr[\textsf{Activity} = a]$, respectively.

To better comprehend the functionality of these factors and promote the discovery of new factors, we propose a causality-based approach to examine the influence of these factors on the utility of learned clauses. Our objective is to use this method to address queries closely tied to the implementation of modern SAT solvers, such as whether a high or low LBD clause offers greater utility and what other factors significantly impact clause utility. The findings can then be leveraged to inform the design of SAT solvers.

Overview

We introduce an approach designed to address SAT-related inquiries. Figure 2 displays our approach's overview, while Algorithm 1 depicts the pseudo-code for our prototype. Initially, we generate observational data from a SAT solver at Line 1. This data records factor values and clause utility (formally defined in Section 3.1). Subsequently, we construct a causal graph representing the causal relationship between all variables, including factors and clause utility, Line 3, based on our prior knowledge and observational data. Meanwhile, we formulate each SAT-related question into a causal query at Line 4. For instance, the LBD-related question can be formulated as a computation of the causal effect from LBD to clause utility. From Lines 5 to 9, we calculate a causal effect estimate. First, we identify a mathematical expression for the estimation known as an *estimand*. Next, the estimand is evaluated on observational data to calculate the causal effect estimate. If the resulting estimate passes the refutation test, it is returned as the final causal effect; otherwise, a failure is reported. Lastly, we return the causal effect corresponding to the original question.

In the subsequent subsections, we will delve into each component depicted in Figure 2. Section 3.1 outlines the process of generating observational data from a SAT solver. Section 3.2 elaborates on the construction of a causal graph using our prior knowledge of SAT solvers and the generated data. Section 3.3 introduces the SAT-related questions of interest and describes their formulation into a causal query. Lastly, Section 3.4 demonstrates the computation of the causal effect for a given query.

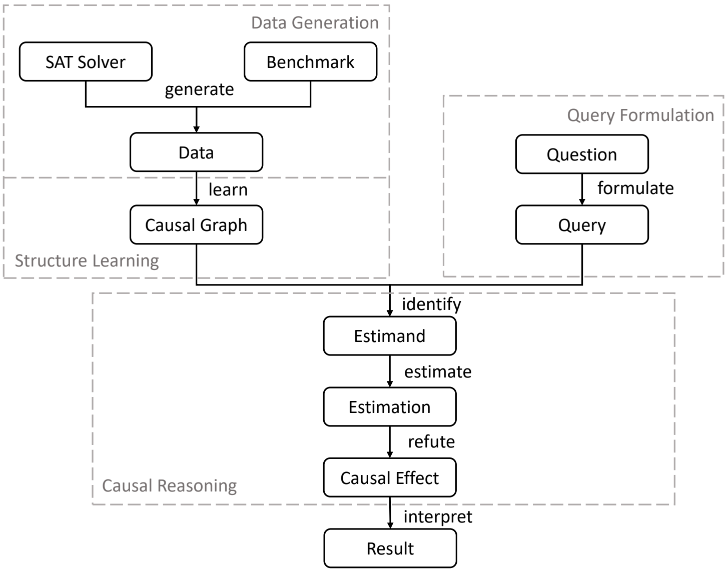


Figure 2 Our approach overview, from data generation to the causal estimate.

Algorithm 1 Our approach (CausalSAT).

```
data ← generate observational data from a SAT solver;
whiteList, blackList ← decode predefined and blocked edges from prior knowledge;
causalGraph ← HillClimbing(whiteList, blackList, data);
query ← formulate a question;
estimand ← Identify(query, causalGraph)
estimates ← Estimate(estimand, causalGraph, data);
passRefutation ← Refute(estimates, query, causalGraph, data);
if passRefutation then return estimates;
else return Failure;
```

3.1 Data Generation

To accomplish our goal of exploring causality and assess the utility of a clause, we collected data that included traces of the SAT solver's execution. These traces captured various characteristics of a clause at different stages of the instance-solving process, as well as the future utility of the clause. We work with a set of unsatisfiable instances. For each instance, we run the SAT solver multiple times with different combinations of heuristics. To complete the data collection procedure, we employ a two-pass system. During the *forward pass*, we run the SAT solver until it reaches the conclusion of unsatisfiability. To gather data on all the learned clauses, we deactivated the learned clause cleaning heuristic during the execution of the SAT solver. Consequently, all the learned clauses were retained throughout the solver's execution. We store data about all the learned clauses at different snapshots of solving. However, this data lacks labeling about whether each clause was useful in proof generation. To address this issue, we execute a *backward pass* using the proof generator DRAT-trim. In this pass, we label all the data collected in the forward pass by how many times the clause will be used in the next 10k conflicts. By combining the data from both passes, we obtain a complete dataset for our study. As a result, each data point in our dataset includes information about the instance, the heuristics used, the learned clauses, some of its features, and the usefulness of each clause in proof generation.

Features and Heuristics. In general, the accuracy of a causal model is often improved with a larger number of features and a larger training dataset. However, many algorithms for detecting causal structures are limited in their ability to handle a large number of features. Therefore, we have selected a subset of important features that are commonly used in modern SAT solver literature. In Table 1, we list all the features we have in our data. In our study, we adopted the definition of clause utility as proposed in [48], which relies on the frequency of clause usage observed in the proofs reconstructed by DRAT-trim [52]. It should be noted that the proof generated by DRAT-trim may differ significantly from the proof generated by the SAT solver, necessitating caution in interpretation.

Table 1 Data collected about clauses during solving.

Branching	Heuristics to determine the order in which variables are assigned values during the search. We consider VSIDS [31] and Maple [24,26].
Restart	Heuristics to determine when the solver should restart. We consider Geometric [50] , LBD-based [4], and Luby [19,27] heuristics.
Size	number of literals in a clause.
LBD	number of distinct decision levels of literals in a clause.
Activity	measure of clause's importance in the search process, based on the clause's involvement in recent conflicts.
UIP	number of times that the clause took part in a 1st-UIP conflict generation since its creation.
Propagation	number of times the clause was used in propagations.
LastTouch	number of conflicts since the clause was used during a 1st-UIP conflict clause generation.
Time	number of conflicts since the generation of this clause.
Utility	within the next 10,000 conflicts, the number of times this clause has been used in DRAT proof generation. The number is weighted based on at which points the clauses are used.

3.2 Structure Learning

A causal graph can be built following our prior knowledge. For example, the clause management heuristic is a known cause for the average LBD of learned clauses. If we preserve the clause of small LBD but remove that of large LBD during clause reduction, the average LBD will go down. Therefore, we can add to the causal graph an edge from clause management heuristic to average LBD to represent the known causality between them. For another instance, both branching and restart heuristics are user-defined parameters, and therefore no other variables inside the SAT solver can cause them. We can then block incoming edges for both heuristics. Overall, we add the following constraints based on our prior knowledge:

- No incoming edges are allowed for Branching and Restart since they are given by users.
- No incoming edges are allowed for Time because time ticks are fixed in our experiments.
- No incoming edges allowed for LBD and Size except for edges from Branching and Restart because LBD and Size are determined before other variables.
- No outgoing edges are allowed for Utility because Utility follows other variables.

However, prior knowledge is insufficient to build a complete causal graph in many scenarios because the knowledge we have about the system is always incomplete, like a SAT solver. Hence, a complete causal graph has to be learned from the observational data. We use the hill-climbing algorithm [44] in our approach to learn a causal graph for its practical performance. The algorithm starts from an initial directed graph. In each step, we randomly add, remove, or reverse an edge of the graph to form candidate graphs. A score function called Bayesian Information Criterion [40], which favors a graph with a simple structure and lower log-likelihood over the data, is then applied to these candidate graphs to select a locally optimal graph. The process continues until a fixed point where we do not find a better graph.

Furthermore, we apply k-fold cross-validation to the learning process to lower the variance of the returned graph. The dataset is initially partitioned into k subsets. Subsequently, for each iteration, we learn a graph on $k-1$ subsets and evaluate its fitness over the remaining subset. Finally, we obtain an averaged graph by taking a majority vote from the k candidates on the existence and the direction of each edge.

There are other structure-learning algorithms available such as the PC Stable algorithm [12]. However, the PC Stable algorithm always returns a graph with a few undirected edges in our experiments, which means the algorithm failed to infer the causal direction between some variables. Due to its algorithmic nature, the hill-climbing algorithm always returns a directed graph, meeting our requirements. A detailed comparison between structure-learning algorithms is out of the scope of this paper, and we refer the interested readers to [41]. Additionally, we present a pseudo-code of the hill-climbing algorithm in Appendix A [54].

3.3 Queries on SAT Solving

Are you curious about whether a clause with a high or low LBD possesses a greater utility? Alternatively, you might be interested in determining which factor – size or LBD – exerts a more significant influence on clause utility. Previously, one would have to rely on personal experience or experiment with heuristics to reach a conclusion. However, with our proposed method, you simply need to transform these inquiries into causal queries, and our causal model will provide the answers.

In this section, we initially present the different query types supported by our approach. Following that, we explore questions closely related to SAT solver implementation and demonstrate the process of formulating them into causal queries. Our primary focus lies on queries concerning clause utility, with the intention of examining the variables that exert a substantial influence on clause utility.

Query Type

Our causal framework accommodates three distinct query categories, as detailed below:

- Average Treatment Effect (ATE) quantifies the variation in the outcome variable when the treatment variable shifts from one value to another.
- Conditional Average Treatment Effect (CATE) encapsulates the ATE, considering a condition for an additional variable.
- We propose the Averaged Conditional Average Treatment Effect (ACATE) concept to account for the need to average the CATE across all values of the conditional variable.

▶ **Definition 3** (Averaged Conditional Average Treatment Effect).

$$
\begin{aligned}
ACATE(X, Y, W, a, b) = \sum_{w} (&\mathbb{E}\left[Y | do(X = a), W = w\right] - \\
&\mathbb{E}\left[Y | do(X = b), W = w\right]) \Pr[W = w]
\end{aligned}
$$

LBD. In modern SAT solvers, LBD serves as a key factor for clause memory management heuristics after being introduced in Glucose [4]. It is assumed that the clause of small LBD has greater utility and is therefore favored during clause reduction. The assumption drives us to ask the question: which clause, with low or high LBD, has greater utility? The question can be translated to calculate the causal effect from LBD to clause utility, corresponding to an ATE query (Q1) in Table 2. The ATE measures the average change of the outcome (clause utility) if the treatment (LBD) changes from one to two. Following the assumption that a clause of small LBD has greater utility, increasing LBD will decrease the utility, and hence the ATE is expected to be negative. If the ATE is indeed negative, we prove the assumption. Otherwise, if ATE is positive, we reach the opposite conclusion that the clause of large LBD has greater utility. Finally, if ATE equals zero, the LBD has no effect on clause utility, and clauses with large or small LBD have the same utility.

Table 2 Causal queries for questions on SAT solving.

	Question	Query
Q1	Which clause, with low or high LBD, has greater utility?	$\text{ATE}(\mathsf{LBD}, \mathsf{Utility}, 2, 1) < 0$
Q2	Which type of clause, large or small, has greater utility? What if the LBD is fixed?	$\begin{cases} \text{ATE}(\mathsf{Size}, \mathsf{Utility}, 2, 1) < 0 \\ \text{ACATE}(\mathsf{Size}, \mathsf{Utility}, \mathsf{LBD}, 2, 1) > 0 \end{cases}$
Q3	Which clause, with low or high LBD, experiences a rapid drop in utility over time?	$\begin{cases} \text{CATE}(\mathsf{Time}, \mathsf{Utility}, \mathsf{LBD} \leq 6, 10000, 0) \geq 0 \\ \text{CATE}(\mathsf{Time}, \mathsf{Utility}, \mathsf{LBD} > 6, 10000, 0) < 0 \end{cases}$
Q4	Which factor, size or LBD, has a greater impact on clause utility?	$\lvert\text{ATE}(\mathsf{Size}, \mathsf{Utility}, 2, 1)\rvert > \lvert\text{ATE}(\mathsf{LBD}, \mathsf{Utility}, 2, 1)\rvert$
Q5	Which factor, besides size, LBD, and activity, has the greatest impact on clause utility?	$\arg\max_{\mathsf{Treatment} \neq \mathsf{Utility}} \{\lvert\text{ATE}(\mathsf{Treatment}, \mathsf{Utility}, 2, 1)\rvert\}$
Q6	Which branching heuristic, VSIDS or Maple, results in a greater clause utility?	$\text{ATE}(\mathsf{Branching}, \mathsf{Utility}, \text{Maple}, \text{VSIDS})$
Q7	Which restart heuristic, Geometric, LBD-based, or Luby, results in the greatest clause utility?	$\begin{cases} \text{ATE}(\mathsf{Restart}, \mathsf{Utility}, \text{Luby}, \text{Geometric}) \\ \text{ATE}(\mathsf{Restart}, \mathsf{Utility}, \text{Luby}, \text{LBD-based}) \\ \text{ATE}(\mathsf{Restart}, \mathsf{Utility}, \text{Geometric}, \text{LBD-based}) \end{cases}$

Size. Modern solvers assume a small clause has higher utility than a large clause, which brings us to the question: which type of clause, large or small, has a greater utility? The question is formulated into the first ATE query (Q2) in Table 2, which implies a decreasing utility when the clause size increases from one to two. If the resulting ATE is negative, a small clause is proved to have greater utility. Otherwise, a large clause has a greater utility if the ATE is positive. Lastly, the size does not affect utility if the ATE is zero.

A historical implementation of Glucose [17] prioritizing large clauses of the same LBD outperformed the one favoring small clauses, which implies that large clauses have greater utility given a fixed LBD. The observation is counter-intuitive because small clauses are usually assumed to be more useful, and people think it still holds when LBD is fixed.

Consequently, we ask the question: which type of clause, large or small, has greater utility given a fixed LBD? The question is formulated into an ACATE query (Q2) in Table 2, which implies a decreasing utility when clause size increases from one to two conditioned on a fixed LBD. The hypothesis will be verified if the ACATE is indeed positive or disproved otherwise.

LBD over Time. Modern SAT solvers adopt multi-tier clause memory management heuristic: more useful clauses are preserved for a long time, while less useful clauses are cached for a short time. Clauses are classified based on LBD. For example, Kissat [8] stores learned clauses with $\mathsf{LBD} \leq 6$ in the long-time tier with the rest in the short-time tier. This observation implies that low-LBD clauses are expected to maintain their utility for a long time, while conversely, the utility of large-LBD clauses is expected to drop quickly over time.

The implication drives us to ask the question: Which clause, with low or high LBD, experiences a rapid drop in utility over time? Following that, we formulate the question into (Q3) in Table 2. The first query formulates that small-LBD clauses do not experience a decrease in utility when time ticks from 0 to 10,000, while the second one encodes that large-LBD clauses experience a utility drop over time. We model the time period as 10,000 conflicts because the data is collected per 10,000 conflicts. If the answers to two queries are both yes, the hypothesis is verified from a causal perspective.

Size vs. LBD. In modern SAT solvers, variants of clause management heuristics are based on size and LBD, but a question remains open: which has a greater impact on clause utility? We formulate the question into (Q4) in Table 2 to compare the causal effects on clause utility between size and LBD. If the size (resp. LBD) has a greater effect on clause utility, we suggest clause size (resp. LBD) be involved more in clause management. Note that we compare the effects over normalized data to eliminate the unfairness due to different orders of magnitude for size and LBD.

Beyond Size, LBD, and Activity. Modern SAT solvers consider size, LBD, and activity as the key factors in predicting clause utility, while other factors received less attention. To motivate new clause management heuristics, we utilize CausalSAT to identify new factors for predicting clause utility. As shown in (Q5) of Table 2, we formulate an optimization query to find the treatment that has the largest effect on clause utility. The discovery may inspire the design of new clause management heuristics.

Branching Heuristics. Modern SAT solvers can only deploy one branching heuristic at a time, which defers the hard choice among heuristics to users. In this work, we provide suggestions from a causal perspective. We ask the question: which branching heuristic results in a greater clause utility? We consider two prevalent heuristics, VSIDS [31] and Maple [24,26]. Following that, we formulated the question into an ATE query (Q6) in Table 2. The query evaluates the effect on clause utility when the branching heuristic changes from VSIDS to Maple. A positive ATE indicates Maple yields a higher utility, while a negative value implies VSIDS demonstrates a superior utility. Otherwise, the ATE equals zero, which implies the branching heuristic has no impact on clause utility.

Restart Heuristics. Considering three dominant restart heuristics Geometric [50], LBD-based [4], and Luby [19,27], we provide suggestions on the choice of restart heuristics by asking the question: Which restart heuristic, Geometric, LBD-based, or Luby, results in the greatest clause utility? Following that, we formulate the question into three ATE queries (Q7) in Table 2, which compares the causal effects on clause utility among three heuristics. For example, if the first two ATEs are positive, Luby leads to higher utility than Geometric and LBD-based. Hence Luby is a preferred heuristic for high clause utility. Similar arguments apply to Geometric and LBD-based heuristics.

3.4 Causal Reasoning

In this section, we outline the process for calculating a query with the aid of a causal graph and observational data. Initially, we demonstrate the identification of controlled variables that bias the effect estimation. Following that, we exemplify the calculation of a causal effect using linear regression and the identified variables. Lastly, we discuss the utilization of refutation tests for validating the employed estimation technique.

3.4.1 Identification

A causal query asks to calculate the effect of treatment on the outcome (with some conditions). The naive way to do the calculation is to intervene in the treatment, i.e., change the treatment from one value to the other and observe the variation in outcome. However, intervening in treatment is infeasible in many applications like SAT solving, where it's impossible to increase the size of a clause and then observe the changes in utility. Alternatively, we can estimate the effect from the observational data. As motivated in Section 2.2, we have to identify a set of *controlled* variables to eliminate bias.

There are three methods to identify the controlled variables: backdoor, frontdoor, and instrumental variable identification. We use backdoor identification in our approach because frontdoor and instrumental variable identification always failed to find these variables in our experiments. Interested readers may refer to [36] for the details of other methods. The backdoor algorithm returns a set of variables called *backdoor set*, which is the smallest set separating the treatment and effect after removing the paths from the treatment to the outcome. These paths carry the pure effect from treatment to the outcome. A backdoor set represents a set of variables that exert an implicit effect on the treatment and outcome, such as the common cause LBD in Figure 1a. Controlling these variables ensures an unbiased estimation of the causal effect. Using a backdoor set, we can convert an ATE query into an expression called *estimand*, which can be evaluated over observational data to estimate the causal effect. Details of the backdoor algorithm are deferred to Appendix B [54].

3.4.2 Effect Estimation

We utilize linear regression as the estimator, given its effectiveness and interpretability. As a widely used estimator for causal effects, linear regression delivers remarkably strong performance. Its estimate is simple to interpret, signifying the average shift in the outcome when the treatment rises by one unit. Consequently, we demonstrate that the coefficient β corresponding to the treatment variable X signifies the ACATE value.

▶ **Lemma 4.** *Suppose $Z \cup W$ is a backdoor set and $\mathbb{E}[Y \mid X = x, Z = z, W = w]$ is linear, i.e., $\mathbb{E}[Y \mid X = x, Z = z, W = w] = \beta_0 + \beta_1 x + \beta_2^T z + \beta_3^T w$. We obtain*

$$ACATE(X, Y, W, a, b) = (a - b)\beta_1$$

We defer the proof to Appendix C [54] due to the page limit.

To identify a backdoor set that incorporates the conditional variable W, we can designate W as an element of the set and seek a valid backdoor set. In practice, we presume $\mathbb{E}[Y \mid X = x, Z = z, W = w]$ to be linear and employ $(a - b)\beta_1$ as an estimate of ACATE.

This conclusion is also applicable to ATE and CATE, with the proof detailed in Appendix [54]. To enhance confidence in statistical estimations, we conduct refutation tests to verify our estimation method. For example, we can assess whether the estimate changes substantially upon adding an independent random variable to the data as a common cause of the treatment and outcome. Interested readers may refer to Appendix D [54] for more details.

4 Experimental Evaluation

We implement a prototype called CausalSAT to evaluate our approach over practical instances. CausalSAT is built on top of a structure-learning package, bnlearn [41], and a causal-reasoning package, DoWhy [43]. We first present CausalSAT fits the dataset and then use it to answer the queries in Table 2.

For our experiments, we used a high-performance computing cluster. We collected traces of the run using instances from the SAT competition benchmarks. Because of the construction of our framework, we could only use unsatisfiable instances. Additionally, the process of collecting traces added significant overhead to the SAT-solving process, preventing us from gathering data from instances that took more than an hour to solve in a modern SAT solver or instances that could be solved within a few seconds. To ensure a variety of data, we chose to use 80 UNSAT instances from the SAT competition benchmarks from 2014-'18. We allowed a 2-hour timeout for all instances to run and record the traces. Afterward, we sampled data from those traces to ensure an equal representation of all types of instances. In total, we gathered 4 million data points. Lastly, we ran the hill-climbing algorithm using 10-fold cross-validation and obtained the causal graph in Figure 3.

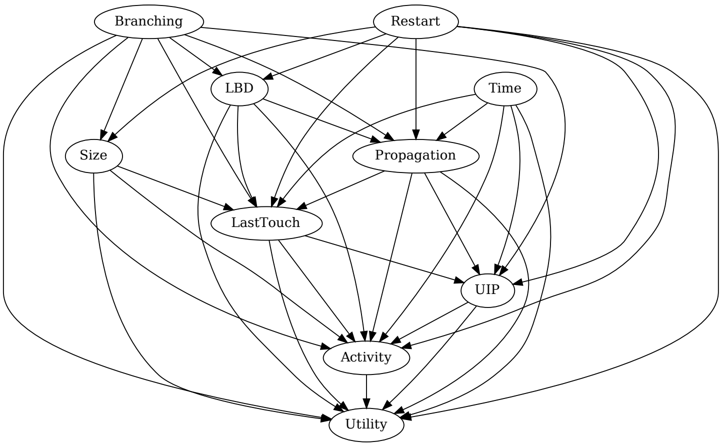


Figure 3 The causal graph CausalSAT derives using 10-fold cross-validation and 4 million points.

Summary. Our model fits the dataset, attaining a comparable loss to both linear models and decision trees while exhibiting a higher Pearson correlation with clause utility than any individual variable. With the model in place, we answer the questions in Table 2. For LBD, a small value leads to a larger clause utility, but large value results in a smaller utility that drops rapidly over time. For size, a large clause has greater utility, and the statement also holds when the LBD is fixed. Overall, the LBD has a greater impact than the size on clause utility. Besides LBD, size, and activity, the number of propagations has the greatest impact. Last but not least, Maple has a greater utility between branching heuristics, and Luby achieves the highest utility among restart heuristics.

4.1 Fitness on Data

We evaluate the fitness of CausalSAT on the dataset by showing a comparable prediction error and a stronger correlation with clause utility than any individual variable. We present the competence of CausalSAT with a linear model and decision tree by evaluating their performance based on the mean squared error (MSE) metric (the lower, the better). We further demonstrate that the Pearson correlation of CausalSAT with clause utility is higher than that of any individual variable.

CausalSAT delivers a comparable prediction error with a linear model and decision tree. We evaluate the mean squared error (MSE) of predicting clause utility for the three models. We adopt 10-fold cross-validation where the dataset is split into ten subsets, and every time the model is trained on nine subsets and evaluated on the remaining subset. The overall MSE is averaged over the 10 evaluations. Table 3 presents the MSE on different data sizes. When 0.5 million data points are available, CausalSAT obtains an MSE of 10050.41, which is smaller than both the linear model and decision tree. The result shows that CausalSAT still fits the data when we don't have much data available. On the other hand, the MSE of the decision tree decreases significantly as the data size increases, and the decision tree finally becomes the best predictor. Note that the prediction is not the focus of a causal model so it is expected to observe a smaller MSE from the decision tree when large data is available.

Table 3 Mean squared error in 10-fold cross-validation.

Data Size	Linear Model	Decision Tree	CausalSAT
0.5 million	10087.79	13400.43	10050.41
1 million	9848.14	10731.84	9810.78
2 million	9986.05	7555.20	9951.33
4 million	9977.85	4335.32	9943.03

Additionally, we evaluate the correlation with Utility for any individual variable and CausalSAT. We compute the Pearson correlation coefficient between any individual variable and Utility except for Branching and Restart. Pearson correlation coefficient is a measure of linear correlation between two variables and has a value between -1 and 1, with 1 (resp. -1) indicating a strong positive (resp. negative) linear relationship and 0 suggesting no linear relationship between the variables. The Pearson correlation for CausalSAT measures the linear relationship between the Utility and the prediction generated by CausalSAT. As shown in Table 4, CausalSAT's prediction has a higher correlation with Utility than any individual variable. Consequently, any individual variable is not a good indicator for Utility while CausalSAT, considering all variables and their causal relationship, delivers a better correlation with clause utility.

Table 4 Correlation with clause utility.

	Activity	LBD	LastTouch	Size	Propagation	UIP	Time	CausalSAT
Correlation	0.2819	-0.0285	-0.0326	-0.0283	0.2381	0.2557	0.0933	0.3425

4.2 Query Answering

We use CausalSAT to answer the questions in Table 2 by calculating the causal effects for the queries. Table 5 lists the calculations and answers, where the causal estimates all pass the three refutation tests. We use the original data to calculate ATE for a single variable due to its interpretability while using normalized data in Q4 and Q5 to compare ATEs between variables to eliminate the unfairness of their different orders of magnitude. We apply standard score normalization to the data. As follows, we explain the answers in detail:

- (A1) LBD has a negative effect on clause utility, indicating that clauses with low LBD have greater utility. Hence, a low-LBD clause should be prioritized over a high-LBD clause in clause memory management.
- (A2) Both ATE and ACATE values are negative, indicating that small clauses have greater utility, which also holds when LBD is fixed, though with a smaller effect. The answer to the fixed-LBD case contradicts the previous observation that a large clause is assumed to have greater utility when LBD is fixed.
- (A3) Time has a positive effect on Utility for a low-LBD clause and a negative effect for a high-LBD clause. The results show that a high-LBD clause experiences a rapid drop in utility while the utility of a low-LBD clause even increases over time. Therefore, low-LBD clauses should be preserved in memory for a longer time than high-LBD clauses.
- (A4) LBD has a larger absolute ATE value than size, indicating a greater impact on clause utility. The result suggests the usage of LBD in a clause memory management heuristic against the size from a causal perspective.
- (A5) Appendix E [54] lists the ATE of all variables on clause utility over the normalized data. Among them, Propagation has the greatest effect. The result suggests the introduction of Propagation into the future design of clause memory management heuristics.
- (A6) Branching has a positive effect on Utility when changing the strategy from VSIDS to Maple. Hence, Maple leads to greater utility and serves as a recommended branching heuristic for high-clause-utility scenarios.
- (A7) Luby restart leads to a greater utility than Geometric and LBD-based, achieving the greatest utility. Hence, Luby is a recommended restart heuristic for high-clause-utility scenarios.

Table 5 Query estimates and their interpretations.

	Query	Answer
Q1	$\text{ATE}(\mathsf{LBD}, \mathsf{Utility}, 2, 1) = -0.2610 < 0$	Low-LBD clause has greater utility.
Q2	$\begin{cases} \text{ATE}(\mathsf{Size}, \mathsf{Utility}, 2, 1) = -0.0314 < 0 \\ \text{ACATE}(\mathsf{Size}, \mathsf{Utility}, \mathsf{LBD}, 2, 1) = -0.0202 < 0 \end{cases}$	Small clause has greater utility, which also holds when LBD is fixed.
Q3	$\begin{cases} \text{CATE}(\mathsf{Time}, \mathsf{Utility}, \mathsf{LBD} \leq 6, 10000, 0) = 0.3772 > 0 \\ \text{CATE}(\mathsf{Time}, \mathsf{Utility}, \mathsf{LBD} > 6, 10000, 0) = -0.0884 < 0 \end{cases}$	High-LBD clause experiences a rapid drop in utility over time.
Q4	$\lvert\text{ATE}(\mathsf{Size}, \mathsf{Utility}, 2, 1)\rvert = 2.6660 < \lvert\text{ATE}(\mathsf{LBD}, \mathsf{Utility}, 2, 1)\rvert = 3.3754$	LBD has a greater impact than size.
Q5	$\arg\max_{\mathsf{Treatment} \neq \mathsf{Utility}} \{\lvert\text{ATE}(\mathsf{Treatment}, \mathsf{Utility}, 2, 1)\rvert\}$	Propagation has the greatest impact on utility.
Q6	$\text{ATE}(\mathsf{Branching}, \mathsf{Utility}, \text{Maple}, \text{VSIDS}) = 18.5745 > 0$	Maple leads to greater utility.
Q7	$\begin{cases} \text{ATE}(\mathsf{Restart}, \mathsf{Utility}, \text{Luby}, \text{Geometric}) = 6.7347 > 0 \\ \text{ATE}(\mathsf{Restart}, \mathsf{Utility}, \text{Luby}, \text{LBD-based}) = 17.7385 > 0 \\ \text{ATE}(\mathsf{Restart}, \mathsf{Utility}, \text{Geometric}, \text{LBD-based}) = 11.0037 > 0 \end{cases}$	Luby leads to the greatest utility.

5 Conclusion

This work serves as the first work to utilize causality to uncover the inner workings of modern SAT solvers. We proposed a framework to calculate the causal effects on clause utility from key factors such as LBD, size, activity, and the choice of heuristics. The causal results verified the "rules of thumb" in solver implementation, for example, that a low-LBD clause is assumed to have greater utility than a high-LBD clause. We also answered questions closely related to SAT solving, for instance, which factor, size or LBD, has a greater impact on clause utility? The causal framework provides a systematical way to quantitatively investigate the relationship among components of a SAT solver, which paves the way to further understanding how modern SAT solvers work.

This work opens several promising directions for future research. Our immediate focus will be enhancing our framework by introducing a new definition for clause utility. This revised definition will consider the frequency of a learned clause's usage by a SAT solver rather than its usage in DRAT proof generation. Additionally, We aim to include a definition for clause utility that encompasses both satisfiable and unsatisfiable instances, expanding beyond our current analysis of unsatisfiable cases. Looking ahead, we intend to incorporate a broader range of features into our study and extend a further investigation into the obtained causal graph's structure and causal estimates' value to enrich our analysis.

References

1 Kartik Ahuja, Yixin Wang, Divyat Mahajan, and Yoshua Bengio. Interventional causal representation learning. In *Proc. of NeurIPS Workshop on Causality for Real-world Impact*, 2022.

2 Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint*, 2019. `arXiv:1907.02893`.

3 Gilles Audemard and Laurent Simon. Glucose: a solver that predicts learnt clauses quality. *SAT Competition*, 2009.

4 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proc. of IJCAI*, 2009.

5 Gilles Audemard and Laurent Simon. On the glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 2018.

6 Armin Biere. Lingeling, Plingeling and Treengeling entering the SAT competition 2013. *Proc. of SAT competition*, 2013.

7 Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proc. of TACAS*, 1999.

8 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In *Proc. of SAT Competition – Solver and Benchmark Descriptions*, 2020.

9 Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In *Proc. of DAC*, 2015.

10 Johann Brehmer, Pim De Haan, Phillip Lippe, and Taco Cohen. Weakly supervised causal representation learning. In *Proc. of UAI Workshop on Causal Representation Learning*, 2022.

11 Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In *Proc. of TACAS*, 2004.

12 Diego Colombo and Marloes H. Maathuis. Order-Independent Constraint-Based Causal Structure Learning. *Journal of Machine Learning Research*, 2014.

13 Stephen A Cook. The complexity of theorem-proving procedures. In *Proc. of STOC*, 1971.

14 Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 1962.

15 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. of SAT*, 2003.

16 Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical cdcl insights from theoretical sat benchmarks. In *IJCAI*, pages 1300–1308, 2018.

17 Laurent Simon Gilles Audemard. Glucose 1.0, 2009. sources/glucose/core/Solver.C:Lines 631–635. URL: `https://www.labri.fr/perso/lsimon/downloads/softwares/glucose_1.0.zip`.

18 Carla P Gomes, Bart Selman, Henry Kautz, et al. Boosting combinatorial search through randomization. *Proc. of AAAI/IAAI*, 1998.

19 Jinbo Huang et al. The Effect of Restarts on the Efficiency of Clause Learning. In *Proc. of IJCAI*, 2007.

20 Yimin Huang and Marco Valtorta. Pearl's calculus of intervention is complete. In *Proc. of UAI*, 2006.

21 Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international SAT solver competitions. *AI Magazine*, 2012.

22 Henry A Kautz, Bart Selman, et al. Planning as Satisfiability. In *Proc. of ECAI*, 1992.

23 Janne I Kokkala and Jakob Nordström. Using resolution proofs to analyse CDCL solvers. In *Proc. of CP*, 2020.

24 Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for SAT solvers. In *Proc. of SAT*, 2016.

25 Jia Hui Liang, Chanseok Oh, Minu Mathew, Ciza Thomas, Chunxiao Li, and Vijay Ganesh. Machine learning-based restart policy for CDCL SAT solvers. In *Proc. of SAT*, 2018.

26 Jia Hui Liang, Pascal Poupart, Krzysztof Czarnecki, and Vijay Ganesh. An empirical study of branching heuristics through the lens of global learning rate. In *Proc. of SAT*, 2017.

27 Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 1993.

28 Mao Luo, Chu-Min Li, Fan Xiao, Felip Manya, and Zhipeng Lü. An effective learnt clause minimization approach for CDCL SAT solvers. In *Proc. of IJCAI*, 2017.

29 Inês Lynce and Joao Marques-Silva. SAT in bioinformatics: Making the case with haplotype inference. In *Proc. of SAT*, 2006.

30 João P Marques-Silva and Karem A Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 1999.

31 Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of DAC*, 2001.

32 Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In *Proc. of SAT*, 2018.

33 Chanseok Oh. Between SAT and UNSAT: the fundamental difference in CDCL SAT. In *Proc. of SAT*, 2015.

34 Chanseok Oh. *Improving SAT solvers by exploiting empirical characteristics of CDCL*. PhD thesis, New York University, 2016.

35 Judea Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 2009.

36 Judea Pearl. *Causality.* Cambridge University Press, 2009.

37 Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, pages 947–1012, 2016.

38 Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proc. of SAT*, 2007.

39 Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proc. of IEEE*, 109(5):612–634, 2021.

40 Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 1978.

41 Marco Scutari. Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software*, 2010.

42 Marco Scutari and Radhakrishnan Nagarajan. Identifying significant edges in graphical models of molecular networks. *Artificial Intelligence in Medicine*, 2013.

43 Amit Sharma, Emre Kiciman, et al. DoWhy: A Python package for causal inference, 2019. URL: `https://github.com/microsoft/dowhy`.

44 Qiang Shen and Rónán Daly. Methods to accelerate the learning of bayesian network structures. In *Proc. of UK Workshop on Computational Intelligence*, 2007.

45 JP Marques Silva and Karem A Sakallah. Conflict analysis in search algorithms for satisfiability. In *Proc. of ICTAI*, 1996.

46 Laurent Simon. Post mortem analysis of sat solver proofs. In *POS@ SAT*, pages 26–40, 2014.

47 Mate Soos, Jo Devriendt, Stephan Gocht, Arijit Shaw, and Kuldeep S Meel. Cryptominisat with CCAnr at the SAT competition 2020. *SAT COMPETITION*, 2020.

48 Mate Soos, Raghav Kulkarni, and Kuldeep S Meel. CrystalBall: Gazing in the Black Box of SAT Solving. In *Proc. of SAT*, 2019.

49 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *Proc. of SAT*, 2009.

50 Toby Walsh. Search in a small world. In *Proc. of IJCAI*, pages 1172–1177, 1999.

51 Ruoyu Wang, Mingyang Yi, Zhitang Chen, and Shengyu Zhu. Out-of-distribution generalization with causal invariant transformations. In *Proc. of CVPR*, pages 375–385, 2022.

52 Nathan Wetzler, Marijn JH Heule, and Warren A Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Proc. of SAT*, 2014.

53 Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of artificial intelligence research*, 2008.

54 Jiong Yang, Arijit Shaw, Teodora Baluta, Mate Soos, and Kuldeep S. Meel. Explaining sat solving using causal reasoning. *arXiv*, 2023. `arXiv:2306.06294`.

LS-DTKMS: A Local Search Algorithm for Diversified Top-k MaxSAT Problem

Junping Zhou
School of Information Science and Technology, Northeast Normal University, Changchun, China

Jiaxin Liang
School of Information Science and Technology, Northeast Normal University, Changchun, China

Minghao Yin[1]
School of Information Science and Technology, Northeast Normal University, Changchun, China
Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China

Bo He
School of Information Science and Technology, Northeast Normal University, Changchun, China

Abstract

The Maximum Satisfiability (MaxSAT), an important optimization problem, has a range of applications, including network routing, planning and scheduling, and combinatorial auctions. Among these applications, one usually benefits from having not just one single solution, but k diverse solutions. Motivated by this, we study an extension of MaxSAT, named Diversified Top-k MaxSAT (DTKMS) problem, which is to find k feasible assignments of a given formula such that each assignment satisfies all hard clauses and all of them together satisfy the maximum number of soft clauses. This paper presents a local search algorithm, LS-DTKMS, for DTKMS problem, which exploits novel scoring functions to select variables and assignments. Experiments demonstrate that LS-DTKMS outperforms the top-k MaxSAT based DTKMS solvers and state-of-the-art solvers for diversified top-k clique problem.

2012 ACM Subject Classification Theory of computation → Random search heuristics; Theory of computation → Constraint and logic programming

Keywords and phrases Top-k, MaxSAT, local search

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.29

Supplementary Material *Dataset*: `https://github.com/LyreRabbit/DTKMS`
archived at `swh:1:dir:d898c27229263839aaeef82ca95094c2f6643f11`

Funding This work is supported by Science and Technology Development Program of Jilin Province (20230101060JC and YDZJ202201ZYTS412), NSFC (under Grant No.61976050, 61972384), the Fundamental Research Funds for the Central Universities 2412019ZD013, and Jilin Science and Technology Department 20200201280JC.

Acknowledgements We would like to thank all the anonymous reviewers for their helpful comments.

1 Introduction

The Maximum Satisfiability (MaxSAT), an optimization version of the famous Satisfiability (SAT) problem, concerns about finding an assignment to satisfy all hard clauses as well as the maximum number of soft clauses by given a general form of propositional formula, which is represented as the Conjunctive Normal Form (CNF) containing both hard and soft clauses. Recently, the success of the MaxSAT algorithms [9, 11, 12, 17, 35] has contributed

[1] Corresponding Author

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 29; pp. 29:1–29:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to a significant number of applications, such as network routing [20], planning and scheduling [27], combinatorial auctions [4], software engineering [13], data analysis [8] and machine learning [19]. Among these real-life applications, one actually benefits from not just one solution but diverse solutions. For example, in a social network, mining diverse communities often helps to find different topics and reduce the amount of overlapping information [7]. In robotic motion planning, practitioners are often interested in finding diverse paths because some pre-computed paths sometimes become invalid due to the relocation of obstacles or the reconfiguration of the robot [31]. In natural language understanding, machine translation systems benefit from working with multiple plausible parses of a sentence because sentences are often ambiguous [15]. In computational biology, computing multiple configurations of a protein structure is believed to be helpful to assess the sensitivity of the model [38]. Thus, in order to better meet the users' needs in numerous applications, it is essential to determine diverse solutions for MaxSAT problem.

This problem is introduced as the Diversified Top-k MaxSAT (DTKMS) problem, which is equivalent to MaxSAT when $k = 1$. Given a CNF formula, the objective of DTKMS is to determine at most k feasible assignments such that each assignment satisfies all hard clauses and the k assignments together satisfy the maximum number of soft clauses. As an extension of the MaxSAT problem, DTKMS is a bi-standard optimization problem that balances correlation and diversity of results. Now, a straightforward method for solving DTKMS involves first exhaustively searching for all feasible assignments and then employing the max k-coverage algorithm. Unfortunately, this method is not practical because the enumeration consumes enormous amounts of time and memory. Thus, to effectively solve DTKMS, two major challenges should be tackled. (1) As the number of feasible assignments for MaxSAT is potentially exponential, how can the diversification requirement be met without generating highly overlapping assignments? (2) Without an exhaustive search, how can the quality of the solutions be guaranteed?

In this work, for replying the above challenges, we propose a local search algorithm for the DTKMS problem, named LS-DTKMS, which features new scoring functions, including *score of variable* and *score of assignment*. To avoid generating highly overlapping assignments, we design a new *score of variable* scheme, unlike the previous one that only works on the current assignment. We evaluate the effect of flipping variables on both the current assignment and the DTKMS solution containing k assignments. By applying the new scheme in our variable selection heuristic, LS-DTKMS is able to generate diversified assignments. In addition, we design a *score of assignment* scheme to estimate the quality of a feasible assignment. After combining it with a key solution updating rule, LS-DTKMS can achieve a guaranteed approximation ratio of 0.25 if the final solution is obtained according to the updating rule. We conduct experiments to compare LS-DTKMS against top-k MaxSAT based DTKMS solvers on top-k MaxSAT instances from the MaxSAT Evaluation (MSE) 2020 top-k track. The results demonstrate that LS-DTKMS significantly outperforms the other solvers. To further verify the effectiveness of our algorithm, we apply LS-DTKMS to Diversified Top-k Clique Search (DTKCS) problem, which is to find k maximal cliques such that they cover the maximum number of vertices in a given graph [37]. The experimental results show that LS-DTKMS has better improvement than the state-of-the-art DTKCS solvers.

The outline of the paper is as follows. We first present some related definitions and a framework of top-k MaxSAT based DTKMS algorithm. In Section 3, we propose a local search algorithm and discuss the details of the techniques involved. In Section 5, we show the experimental results. Finally, we conclude the paper.

2 Diversified Top-k MaxSAT Problem

2.1 Preliminaries

A literal is either a Boolean variable x (positive literal) or its negation $\neg x$ (negative literal). The polarity of a positive literal is 1, while the polarity of a negative literal is 0. A clause is a disjunction of literals. A formula F in Conjunctive Normal Form (CNF) is a conjunction of clauses, which can be represented as a set of clauses. Any variable in F may take values *true* or *false*. An assignment α_i for F with n variables, $x_1, x_2, ..., x_n$, is a mapping that assigns each variable a value and it is expressed by $\alpha_i = v_1 v_2 ... v_n$, where v_i $(1 \leq i \leq n)$ is the corresponding value of x_i. Given an assignment, a clause is satisfied *iff* at least one of its literals is *true*, and a formula F is satisfied *iff* each clause in F is satisfied.

The Maximum Satisfiability (MaxSAT) problem is to find an assignment that satisfies all hard clauses and maximizes the number of satisfied soft clauses of a given CNF formula, $F = \mathit{Hard} \cup \mathit{Soft}$, whose clauses can be distinguished into hard clauses and soft clauses, and *Hard* (resp. *Soft*) represents the set of hard (resp. soft) clauses. For a MaxSAT formula F, we say an assignment α is a feasible assignment of F *iff* it satisfies all hard clauses of F. Given a set of feasible assignments, $S = \{\alpha_1, \alpha_2, \cdots, \alpha_k\}$, of a MaxSAT formula F, the private satisfied soft clauses of α_i $(\alpha_i \in S)$, denoted by $\mathit{priv}(\alpha_i, S)$, is the subset of soft clauses only satisfied by α_i, i.e., $\mathit{priv}(\alpha_i, S) = \mathit{sat}(\alpha_i) \setminus \mathit{sat}(S \setminus \{\alpha_i\})$, where $\mathit{sat}(\alpha_i)$ is the set of satisfied soft clauses by α_i.

▶ **Definition 1** (Diversified Top-k MaxSAT (DTKMS) Problem). *Given a formula $F = \mathit{Hard} \cup \mathit{Soft}$ and a positive integer k, the Diversified Top-k MaxSAT problem is to compute a set S with at most k feasible assignments such that these feasible assignments in S satisfy the maximum number of soft clauses of F; that is to say, $|\mathit{sat}(\alpha_1) \cup \mathit{sat}(\alpha_2) \cup \cdots \cup \mathit{sat}(\alpha_k)|$ is maximized, where $\mathit{sat}(\alpha_i)$ $(\alpha_i \in S$ and $1 \leq i \leq k)$ is the set of soft clauses satisfied by α_i.*

It is easy to see that the DTKMS problem is a generalization of MaxSAT, where MaxSAT aims to find one solution, while DTKMS attempts to find k diversified solutions. Since both MaxSAT and DTKMS study the same type of formulae, which involve hard and soft clauses, to avoid confusion, we will distinguish the formulae according to different problems, expressed as DTKMS or MaxSAT formulae. Given a DTKMS formula F and an integer k, we use S to denote the solution of a DTKMS instance F and $\mathit{sat}(S)$ to denote the set of soft clauses satisfied by S.

▶ **Definition 2** (Hamming Distance). *Given two feasible assignments α_i and α_j of a DTKMS formula, the hamming distance between α_i and α_j, denoted by $\mathit{HDist}(\alpha_i, \alpha_j)$, is the number of variables whose corresponding values in the two feasible assignments are different.*

For example, let $\alpha_1 = 110$ and $\alpha_2 = 100$ be two feasible assignments of a formula with three variables x_1, x_2 and x_3. There is only one variable x_2 with different values in α_1 and α_2, so the hamming distance between the two feasible assignments is 1. Clearly, for two assignments α_i and α_j, they are not identical if $\mathit{HDist}(\alpha_i, \alpha_j) > 0$. The higher hamming distance, the higher degree of diversification of two feasible assignments. Note that the hamming distance $\mathit{HDist}(\alpha_i, \alpha_j)$ can be calculated in polynomial time [30]. Given a set of feasible assignments S and a feasible assignment $\alpha \notin S$, we use $\mathit{HDist}(\alpha, S) > 0$ to denote α is different from any feasible assignment in S, i.e., for $\forall \alpha_i \in S$, we have $\mathit{HDist}(\alpha, \alpha_i) > 0$.

Algorithm 1 LS-DTKMS.

```
   Input: a DTKMS instance $F = Hard \cup Soft$, an integer $k$, and a cutoff time
   Output: a solution of DTKMS $S^*$
1  $m = m_0$, $S^* = \emptyset$;
2  while elapsed time < cutoff do
       // $m_{max}$ and $m_0$ are parameters
3      if $m < m_{max}$ then $m \leftarrow 2 \times m$;
4      else $m_0 \leftarrow m_0 + 1$, $m \leftarrow m_0$;
5      for each soft clause $c$ do
6          $ClauseConf[c] = 1$
7      $S \leftarrow$ ConstructS$(F, m)$;
8      if $|sat(S)| = |Soft|$ then return $S$;
9      $S \leftarrow$ UpdateS$(F, S, m)$;
10     if $|sat(S)| > |sat(S^*)|$ then $S^* \leftarrow S$;
11 return $S^*$;
```

2.2 Top-*k* MaxSAT based DTKMS Algorithm

This subsection presents a top-k MaxSAT based DTKMS algorithm. This algorithm follows a greedy algorithm for the max k-coverage problem, which is the problem of selecting k subsets from a collection of subsets such that their union contains as many elements as possible. Given a DTKMS instance F and an integer k, this algorithm iteratively selects the best feasible assignment that can satisfy the maximum number of soft clauses and then adds it into the solution S until the top-k feasible assignments are added into S. In essence, the whole process can be accomplished by a top-k MaxSAT solver, which outputs the top-k feasible assignments. This algorithm can achieve an approximation ration of 0.632, which is the best-possible polynomial time approximation algorithm for the k-coverage problem [29].

3 A New Local Search Algorithm for DTKMS

This section describes our local search algorithm, called LS-DTKMS, for solving DTKMS on top level. Details of important components will be presented in the following.

3.1 Framework of LS-DTKMS

LS-DTKMS (Algorithm 1) alternatively performs solution construction (*ConstructS*) and solution update (*UpdateS*) until a given cutoff time is reached. At first, LS-DTKMS initializes S^* and m (line 1), where m is a parameter used in the Best From Multiple Selections (BMS) strategy [10]. The BMS strategy is a probabilistic method for choosing a variable of good quality from a large set, which is effective for improving the performance of local search. Specifically, the strategy chooses m random variables, and returns the best one. Then the algorithm enters a loop (lines 2–10). In each iteration, the parameter m is changed to obtain diversified results (lines 3–4), and *ClauseConf* (described in the next subsection) is initialized for each soft clause (lines 5–6). Thereafter a solution S with at most k feasible assignments is constructed (line 7). If all soft clauses are satisfied by S, S is directly returned (line 8); otherwise, *UpdateS* is performed to improve the quality of S (line 9). If the new solution obtained by *UpdateS* is better than the best-found solution S^*, S^* is updated. Finally, when the loop terminates, LS-DTKMS returns S^* (line 11).

3.2 Finding a Single Feasible Assignment with Diversity

Local search for DTKMS requires that the k generated feasible assignments are lowly overlapping with each other so as to enhance the usefulness of each individual feasible assignment. Specifically, when a feasible assignment has been built, DTKMS problem prefers the next feasible assignment that is dissimilar with the old one. Thus, it is vital to generate diversified feasible assignments to be better applied in both *ConstructS* and *UpdateS*. To get such feasible assignments, we define a novel score of variable and a variable selection heuristic.

3.2.1 Score of Variable

The variable scoring function is defined by incorporating the benefit of the number of satisfied soft clauses under the candidate solution S and the increment of total weight of satisfied clauses under the current assignment by flipping a variable. The specific definition is as follows.

▶ **Definition 3** (score of variable). *Given a DTKMS formula F, a candidate solution S containing at most k assignments, and an assignment $\alpha \in S$, the score of a variable v, denoted by $score(v)$, is defined as $score(v) = \lambda_1 \cdot score_1(v) + \lambda_2 \cdot score_2(v)$, where λ_1, λ_2 $(0 \leq \lambda_1, \lambda_2 \leq 1)$ are weighting factors, and $score_1(v)$ and $score_2(v)$ are defined as follows.*

- *$score_1(v)$ is defined under the candidate solution S. The $score_1$ of v is $score_1(v) = |sat((S \setminus \{\alpha\}) \cup \{\alpha'\})| - |sat(S)|$, where α' is an assignment obtained by flipping the value of the variable v in α.*
- *$score_2(v)$ is defined under the current assignment. The $score_2$ of v is defined as $score_2(v) = make(v) - break(v)$, where $make(v)$ and $break(v)$ represent the total weights of the clauses which would become satisfied and falsified under α respectively if the value of v is flipped.*

The weights of clauses are set using the clause weighting mechanism [21], which works as follows. At first, the weight of each hard and soft clause is set to 1. Then, the update rules involve: (1) With probability sp, for each satisfied hard clause c with $w(c) > 1$, $w(c) = w(c) - h\_inc$, and for each satisfied soft clause c with $w(c) > 1$, $w(c) = w(c) - 1$; (2) With probability $1 - sp$, for each falsified hard clause c, $w(c) = w(c) + h\_inc$, and for each falsified soft clause c with $w(c) < weightLimit$, $w(c) = w(c) + 1$. In the above rules, h_inc $(h\_inc > 1)$ is a constant, and $weightLimit$ is a limit of the maximum weight value of soft clauses.

Apparently, $score_1(v)$ and $score_2(v)$ both encourage the transformation of the clauses from falsified to satisfied. The $score_1(v)$ inclines the contribution of soft clauses from a global perspective. When flipping variables with $score_1(v)$, the variables with the greater increment of the number of satisfied soft clauses under the candidate solution may be chosen. Flipping such variables is beneficial to finding a better solution. The $score_2(v)$ focuses on emphasizing the importance of hard clauses from a local point of view. When flipping variables using $score_2(v)$, it is likely that the picked variables have greater increment of the weight of satisfied hard clauses under the current assignment. Flipping such variables contributes to finding a better assignment. By integrating $score_1(v)$ and $score_2(v)$, our scoring function is more flexible in the sense that it can select variables according to both local and global perspectives.

3.2.2 Variable Selection Heuristic

In this part, two variable selection heuristics are designed to increase the diversity of the search. One is used in initial assignment construction to generate a diversified initial assignment. The other is applied in local search to modify the initial assignment with the aim of finding a feasible assignment.

The first variable heuristic strategy is implemented by using a Boolean array *ClauseConf* to express the state of each soft clause, where $\mathit{ClauseConf}[c] = 1$ identifies the soft clause c is falsified and unselected during each loop iteration in LS-DTKMS; otherwise, $\mathit{ClauseConf}[c] = 0$. We prefer to choose a variable in the clause c with $\mathit{ClauseConf}[c] = 1$. The reason for doing so is that we prefer to choose the falsified soft clauses to satisfy as many soft clauses in $\mathit{Soft}\setminus\ \mathit{sat}(S)$ as possible. The details of the update rules for *ClauseConf* are given as follows. At first, the *ClauseConf* of each soft clause is initialized to 1. Then *ClauseConf* is updated based on two circumstances.

1. During finding a feasible assignment process: when a soft clause c is selected during one iteration (*ConstructS* and *UpdateS* execution) in LS-DTKMS, then $\mathit{ClauseConf}[c] = 0$.
2. During updating the solution S process: when removing α from S, the *ClauseConf* of each clause c in $\mathit{priv}(\alpha, S)$ is set to 1; when adding α into S, the *ClauseConf* of each clause c in $\mathit{priv}(\alpha, S)$ is set to 0.

After a clause c is picked, a variable is selected randomly from c to flip so as to build a initial assignment dissimilar with the one extracted from the candidate solution S.

The second variable selection heuristic is a two-priority-level heuristic with the purpose of constructing a diversified feasible assignment, which works as follows.

1. The first priority level: If there exist variables whose $\mathit{score}(v) > 0$, choose a variable with BMS strategy, breaking ties by selecting the one that is least recently flipped. In details, if the number of the variables with $\mathit{score}(v) > 0$ is less than a constant m, choose a variable with the greatest *score*; otherwise, a set is built by randomly selecting m variables whose $\mathit{score}(v) > 0$ and then a variable with the highest *score* in the newly-built set is selected.
2. The second priority level: There are no variables with $\mathit{score}(v) > 0$, which indicates that the local search is stuck in the local optimum. In this case, the weights of the clauses are first updated. Then a random falsified hard clause is selected if such clauses exist; otherwise, a random falsified soft clause is selected. Finally, a variable with the highest *score* is chosen from the clause, breaking ties by selecting the one that is least recently flipped.

3.2.3 The Framework of Finding a Single Feasible Assignment

The pseudo code of finding a single feasible assignment *FindAssignment* is outlined in Algorithm 2. *FindAssignment* consists of two phases: initial assignment construction phase (lines 2–13) and local search phase (lines 14–25). In the first phase, with a certain probability p (the noise parameter), an assignment is randomly generated (lines 2–4). With a certain probability $1 - p$, the last feasible assignment is extracted from S. Then a variable v is selected using the first variable heuristic strategy and the initial assignment is generated by flipping the selected variable v (lines 6–13). After building an initial assignment α, *FindAssignment* enters the local search phase, which iteratively modifies α until a given time limit is reached. During each iteration, if a new feasible assignment is constructed, which uses the hamming distance to measure, the assignment is returned (lines 15–16). Otherwise, a variable is selected through the two-priority-level heuristic to find a new assignment (lines 17–25). Finally, the assignment α^* is returned (line 26).

Algorithm 2 FindAssignment(F, S, m).

```
1  $\alpha^* = \emptyset$;
   // initial assignment construction phase
2  if with probability $p$ then
3      $\alpha \leftarrow$ generate a random assignment;
4      update the weights of clauses;
5  else
6      $\alpha \leftarrow$ extract the last feasible assignment from $S$;
7      $S \leftarrow S \setminus \alpha$;
8      if $C = \{c | ClauseConf\,[c] = 1\} \neq \emptyset$ then
9          $c \leftarrow$ randomly select a falsified soft clause under $\alpha$ in $C$;
10     else
11         $c \leftarrow$ randomly select a falsified soft clause in $Soft \setminus sat(S)$;
12     $v \leftarrow$ select a variable from $c$ with the highest score;
13     $\alpha \leftarrow$ flip $v$ in $\alpha$;
   // local search phase
14 while elapsed time $<$ cutoff do
15     if $\alpha$ is a feasible assignment and $HDist(\alpha, S) > 0$ then
16         $\alpha^* \leftarrow \alpha$; break;
17     if $D = \{v | score(v) > 0\} \neq \emptyset$ then
18         $v \leftarrow$ select a variable from $D$ with BMS strategy;
19     else
20         update the weights of clauses;
21         if exists falsified hard clauses then
22             $c \leftarrow$ select a hard clause randomly;
23         else $c \leftarrow$ select a soft clause randomly;
24         $v \leftarrow$ select a variable from $c$ with the highest score;
25         $\alpha \leftarrow$ flip $v$ in $\alpha$;
26 return $\alpha^*$;
```

3.3 Constructing a Candidate Solution for DTKMS Problem

In the subsection, we present the solution construction algorithm *ConstructS* in Algorithm 3 to build a candidate solution S with at most k feasible assignments. In *ConstructS*, if the size of S is less than k or a given time limit is not reached, a loop is executed iteratively (lines 2–6). In each iteration, *FindAssignment* is called to individually generate a diversified feasible assignment (line 4). If the new assignment is not empty, it is inserted into S (line 5). Next, *ClauseConf* is updated according to the update rules (line 6). Note that when a candidate solution has been generated by *ConstructS*, the feasible assignments in S are different, which can be guaranteed by *FindAssignment*.

3.4 Solution Updating for DTKMS Problem

When a candidate solution has been generated by *ConstructS*, our algorithm needs to further improve the quality of the solution, and this leaves a question: how to evaluate the quality of a feasible assignment in a solution. In the following, we define a scoring function on assignments to evaluate the importance of each feasible assignment.

Algorithm 3 ConstructS (F, S, m).

```
S = ∅, α = ∅;
while |S| < k or elapsed time < cutoff do
    if |sat(S)| = |Soft| then break;
    α ← FindAssignment(F, S, m);
    if α ≠ ∅ then S ← S ∪ {α};
    update ClauseConf;
return S
```

3.4.1 Score of Assignment

The scoring function on assignments is crucial in the local search algorithm for DTKMS because it provides the measure of each assignment in a solution, which helps to decide how to replace an old assignment from a solution and thus further improve the solution.

▶ **Definition 4 (score of assignment).** *Given a solution S of a DTKMS instance, and an assignment $\alpha \in S$, the score of α is $score(\alpha) = |priv(\alpha, S)|$.*

By using the *score of assignment* scheme, a solution updating rule works as follows by given a a solution S of a DTKMS instance, and any an assignment $\alpha \in S$.

- When an assignment needs to be removed from S, we pick the one with the lowest $score(\alpha)$. Note that such an assignment may not be the one with the least number of satisfied soft clauses. Our algorithm LS-DTKMS prefers removing an assignment α_{min} with the lowest $score(\alpha)$ because it does very little contribution to the DTKMS solution, even if it may satisfy a large number of soft clauses.
- After removing an assignment α_{min} with the lowest $score(\alpha)$ from S based on the above method, we construct a new solution S' by adding an assignment α. To guarantee the quality of the solution, the assignment α to be added should hold the Inequality (1).

$$|priv(\alpha, S')| > |priv(\alpha_{min}, S)| + \frac{|sat(S)|}{|S|} \tag{1}$$

where $S' = S \setminus \alpha_{min} \cup \{\alpha\}$ and $|S|$ records the number of assignments in the solution S. With the help of the above solution updating rule, we can guarantee that if a solution is obtained according to Inequality (1), then our algorithm LS-DTKMS can achieve a guaranteed approximation ratio of 0.25, which is demonstrated in Lemma 5.

▶ **Lemma 5.** *Let F be a DTKMS instance, k be an integer, S^* be an optimal solution of F, and S' be a solution updated by Inequality (1). Then $|sat(S')| \geq 0.25 \times |sat(S^*)|$.*

Proof. When an algorithm for DTKMS constructs a solution S'_i from S'_{i-1}, the condition $|priv(\alpha, S'_i)| > |priv(\alpha_{min}, S'_{i-1})| + \frac{|sat(S'_{i-1})|}{|S'_{i-1}|}$ is equivalent to $|E|+|G| > |C|+|E|+\frac{|sat(S'_{i-1})|}{|S'_{i-1}|}$ represented in Figure 1. Since $|sat(S'_{i-1})| = |A| + |B| + |C| + |D| + |E| + |F|$, $|sat(S'_i)| = |A| + |B| + |D| + |E| + |F| + |G|$, and $|S'_{i-1}| = k$, we have $|sat(S'_i)| > (1 + \frac{1}{k})|sat(S'_{i-1})|$.

Next, we prove the fact that solving the DTKMS problem is equivalent to the max k-coverage. We reduce the max k-coverage problem to the DTKMS problem as follows.

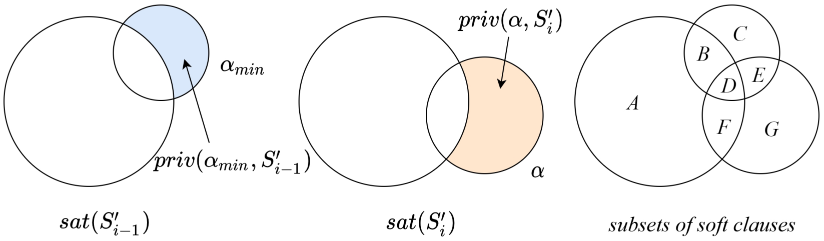


Figure 1 The illustration of Lemma 5.

1. For each element u_i in U, create a variable x_i.
2. For each element u_p in the subset C_i and each element u_q in the subset $U \setminus C_i$, create a hard clause $\neg x_p \vee \neg x_q$.
3. For each element u_i in U, create a soft clause x_i.

We can draw a conclusion that solving the DTKMS problem is equivalent to the max k-coverage problem, which is to select k subsets from $C = \{C_1, C_2, \ldots, C_m\}$ such that their union has the maximum cardinality by giving a set U of n elements and an integer k. In addition, we can obtain the Inequality (2) $|cov(C'_i)| > (1 + \frac{1}{k})|cov(C'_{i-1})|$, where $cov(C'_i)$ is the set of elements covered by a solution C'_i of a max k-coverage instance. When Inequality (2) is satisfied, a theoretical result that a max k-coverage algorithm can achieve a guarantee approximation ration of 0.25 is proved in [5]. Therefore, we can conclude that $|sat(S')| \geq 0.25 \times |sat(S^*)|$. ◀

3.4.2 The Framework of Solution Updating

The framework of solution updating algorithm *UpdateS* is presented in Algorithm 4. At first, *UpdateS* initializes the solution S' and *step* (line 1). Then it enters a loop (lines 2–10), in which *UpdateS* iteratively generates a feasible assignment and updates S with the solution updating rule described in the above. Significantly, if a solution is obtained by Inequality (1), our algorithm can achieve a guaranteed approximation ratio of 0.25. Finally, a better solution is returned (line 11).

Algorithm 4 UpdateS (F, S, m).

```
1  $step = 0$, $S' = \emptyset$;
2  while within the time limit do
3      if $step > maxstep$ then break;
4      $\alpha \leftarrow$ FindAssignment$(F, S, m)$;
5      $\alpha_{min} \leftarrow$ select an assignment $\alpha$ with the lowest $score(\alpha)$ from $S$;
6      $S' = S \setminus \alpha_{min} \cup \{\alpha\}$;
7      if $|priv(\alpha, S')| > |priv(\alpha_{min}, S)| + \frac{|sat(S)|}{|S|}$ then
8          $S \leftarrow S'$, $step \leftarrow 0$;
9      else $step \leftarrow step + 1$;
10     update $ClauseConf$;
11 return $S$;
```

Table 1 Description of the top-k track benchmarks in MSE 2020.

Families	$\|Var\|$	$\|Hard\|$	$\|Soft\|$	Families	$\|Var\|$	$\|Hard\|$	$\|Soft\|$
aes (1)	147	240	147	maxone (2)	485	2817	485
aes-key-recovery (1)	21368	372240	407	MaxSATQueriesinInterpretableClassifiers (5)	1991	14448	1058
atcoss (2)	192216	890542	301	mbd (2)	29703	73188	4637
bcp (2)	152	462	125	packup (5)	13138	73769	7572
CircuitDebuggingProblems (2)	144580	0	432737	protein_ins (5)	2253	2190700	59
CircuitTraceCompaction (2)	158890	494821	20	pseudoBoolean (3)	839	1610	815
close_solutions (4)	68053	2600438	67868	railway-transport (2)	64743	1668566	4945
ConsistentQueryAnswering (3)	44526	46593	10856	ramsey (1)	36	0	210
des (3)	86878	388071	4707	reversi (6)	2981	16521	45
drmx-atmostk (3)	927	1408	47	scheduling (1)	201204	767081	1707
fault-diagnosis (1)	137900	758138	49770	SeanSafarpour (1)	238290	0	936006
frb (1)	760	41367	760	treewidth-computation (3)	88809	814018	60
gen-hyper-tw (1)	62531	197071	48	uaq (3)	2205	3805	189
maxclique (3)	182	16885	182	xai-mindset2 (1)	1330	4763	378
MaximumCommonSub-GraphExtraction (3)	2044	99615	41				

4 Experimental Evaluation

In this section, we carry out two experiments to evaluate the performance of LS-DTKMS. The first experiment compares LS-DTKMS against top-k MaxSAT based DTKMS algorithms on top-k maxsat instances from the MaxSAT Evaluation (MSE) 2020 top-k track[2]. The second experiment applies LS-DTKMS to Diversified Top-k Clique Search (DTKCS) problem, whose instances are from DIMACS graph benchmarks[3].

4.1 Experimental Preliminaries

Our algorithm LS-DTKMS is implemented in C++ and compiled by g++ with "-o3". There are 9 parameters in it. (1) m_0, m_{max}: the initial and the maximum value of the number of samplings used in BMS, respectively. (2) p: the probability, used to generate an initial assignment. (3) sp: the smooth probability, used in the clause weighting scheme. (4) h_{inc}: the increment of falsified hard clauses, used in the clause weighting scheme. (5) $WeightLimit$: the limit on soft clause weight, used in the clause weighting scheme. (6) $maxstep$: the limit on the step of updating the solution. (7) λ_1, λ_2: two coefficients of $score(v)$, used for variable selection. The parameters are tuned according to our experience, and are listed as follows: $m_0 = 15$, $m_{max} = 125$, $p = 0.2$, $sp = 0.01$, $h_{inc} = 2$, $weightLimit = 1$, $maxstep = 23\times10^6$, $\lambda_1 = 0.6$, and $\lambda_2 = 0.4$.

The first experiment compares LS-DTKMS with four top-k MaxSAT based DTKMS algorithms, all of which exploit top-k MaxSAT solvers from top-k track in MSE 2020. The solvers in the top-k track can return k best feasible assignments, which follows the idea of the DTKMS algorithm based on top-k MaxSAT. In the experiment, we use four types of top-k MaxSAT solvers: MaxHS [6], Open-WBO [26], Maxino[3], and RC2 [18]. In each type, we exploit the best one, namely, MaxHS, Open-WBO, maxino, and RC2-A, where RC2-A is the best top-k MaxSAT solver by far. In the second experiment, we evaluate LS-DTKMS with two state-of-the-art incomplete DTKCS solvers, TOPKLS [32] and HEA-D [34]. TOPKLS and HEA-D are state-of-the-art incomplete solvers for DTKCS problem, where HEA-D is the best one by far. In addition, to ensure LS-DTKMS can solve DTKCS benchmarks, we model these instances into DTKMS instances using usual encoding [22]. All experiments are conducted on a server with an Intel(R) Xeon(R) 2.10GHz CPU and 256GB of memory under CentOS Linux release 7.9.2009. For each instance, four top-k MaxSAT based DTKMS

[2] https://maxsat-evaluations.github.io/2020/benchmarks.html

[3] https://iridia.ulb.ac.be/~fmascia/maximum_clique/

Table 2 Comparison on top-k maxsat with k=2 and k=3.

	k=2					k=3				
Families	Open-WBO	MaxHS	Maxino	RC2-A	LS-DTKMS	Open-WBO	MaxHS	Maxino	RC2-A	LS-DTKMS
aes (1)	NA(600.00)	129(25.34)	130(**0.23**)	130(0.20)	**147**(7.38)	NA(600.00)	132(25.34)	132(**0.23**)	132(0.19)	**147**(8.37)
aes-key-recovery (1)	NA(600.00)	406(600.00)	NA(600.00)	406(600.00)	**407**(**40.92**)	NA(600.00)	406(600.00)	NA(600.00)	406(600.00)	**407**(**62.34**)
atcoss (2)	265.5(364.80)	265.5(**23.54**)	265.5(25.87)	265.5(43.51)	**290.5**(53.49)	265.5(102.71)	265.5(**23.96**)	265.5(26.24)	265.5(43.83)	**290.5**(56.31)
bcp (2)	73.5(0.02)	73.5(0.01)	73.5(0.01)	73.5(**0.00**)	**74**(7.43)	73.5(0.02)	73.5(**0.01**)	73(**0.01**)	73.5(**0.01**)	**74**(7.93)
CircuitDebugging-Problems (2)	432736(322.08)	432736(299.53)	432736(**4.09**)	432736(4.31)	**432737**(23.71)	432736(322.25)	**432736.5**(299.76)	432736(4.22)	432736(**6.41**)	**432736.5**(24.69)
CircuitTraceCompaction (2)	12.5(22.60)	12(37.54)	12.5(**9.52**)	12.5(20.99)	**14**(13.61)	13(22.02)	13(38.90)	13(9.58)	13(20.28)	**16.5**(**11.55**)
close_solutions (4)	67857(44.56)	67857(178.48)	67857.3(12.45)	67857(**2.45**)	**67861.3**(28.55)	67858.3(43.71)	67858.3(158.82)	67858.3(12.46)	67858.3(**2.92**)	**67862**(31.66)
ConsistentQuery-Answering (3)	**0**(1.63)	**0**(**0.05**)	**0**(0.25)	**0**(0.05)	**0**(21.42)	**0**(1.62)	**0**(**0.06**)	**0**(0.25)	**0**(**0.06**)	**0**(23.06)
des (3)	3978(251.09)	3977(333.09)	4699.3(112.24)	4700(**30.56**)	**4701.7**(103.27)	3978(251.61)	3978(251.17)	4699.3(112.79)	4700(**30.25**)	**4701.7**(61.81)
drmx-atmostk (3)	29(0.67)	29(98.53)	19(201.58)	28.7(21.05)	**32**(132.99)	29.3(**0.68**)	30(94.93)	19(201.60)	29.7(21.33)	**44**(125.54)
fault-diagnosis (1)	**49593**(34.63)	**49593**(122.39)	**49593**(24.46)	**49593**(**7.01**)	**49593**(8.22)	**49593**(38.30)	**49593**(150.28)	**49593**(25.17)	**49593**(**7.39**)	**49593**(8.43)
frb (1)	40(46.34)	NA(600.00)	NA(600.00)	42(**6.97**)	**54**(84.00)	40(46.56)	NA(600.00)	NA(600.00)	43(**7.14**)	**77**(89.27)
gen-hyper-tw (1)	**44**(128.45)	**44**(**117.65**)	**44**(233.96)	**44**(117.94)	**44**(121.98)	**44**(131.84)	**44**(61.67)	**44**(242.70)	**44**(**116.78**)	**44**(127.47)
maxclique (3)	18.7(1.37)	20(56.54)	19.7(4.91)	19.3(**0.56**)	**30**(11.58)	116.3(1.39)	116(61.49)	116.3(4.96)	116.3(**0.00**)	**135.7**(10.55)
MaximumCommonSub-Graph-Extraction (3)	19(226.61)	20.3(130.08)	20.3(122.02)	20(**107.98**)	**22.3**(149.94)	20(224.80)	21.3(218.50)	21.3(120.73)	21(**105.84**)	**25**(152.61)
maxone (2)	238.5(0.18)	238(0.18)	238.5(**0.05**)	238(0.21)	**254**(79.91)	245(0.18)	240(0.18)	240(**0.05**)	238(0.23)	**257.5**(73.90)
MaxSATQueriesinInterpretable-Classifiers (5)	527.2(23.11)	553.3(89.70)	525.6(**1.98**)	526.6(6.21)	**556**(68.68)	528.2(23.48)	554.5(120.08)	526.6(**1.98**)	528.2(6.62)	**557.2**(69.06)
mbd (2)	4616(198.04)	4616(61.28)	4615.5(1.70)	4616(0.59)	**4636**(**0.17**)	4617(194.40)	4617(58.63)	4616.5(1.70)	4617(0.61)	**4636**(**0.19**)
packup (5)	6959.8(11.81)	6957.4(6.58)	6958.2(1.03)	6959.2(**0.78**)	**7209.8**(25.10)	6965.4(11.98)	6966.6(6.88)	6966(1.03)	6964.8(**0.91**)	**7209.8**(29.17)
protein_ins (5)	29.6(85.07)	30.4(98.79)	30(206.86)	29.6(**33.40**)	**38.2**(140.30)	30.6(88.13)	31(192.14)	30.6(188.86)	29.8(**33.89**)	**39.8**(104.50)
pseudoBoolean (3)	408(0.81)	408(0.05)	407.7(0.07)	408(**0.04**)	**409**(1.70)	408(0.78)	408.3(0.05)	407.7(0.07)	408(**0.05**)	**410**(2.23)
railway-transport (2)	4920.5(27.41)	4921(44.00)	4920.5(6.67)	4920.5(**4.36**)	**4932.5**(14.94)	4775.5(27.95)	4775.5(34.61)	4775.5(6.71)	4775.5(**4.43**)	**4932.5**(19.35)
ramsey (1)	**210**(**0.10**)	**210**(0.65)	**210**(0.72)	**210**(0.31)	**210**(16.27)	**210**(**0.10**)	**210**(0.66)	**210**(0.69)	**210**(0.32)	**210**(16.32)
reversi (6)	32.3(**2.01**)	31.8(10.50)	31.8(6.52)	32.17(4.97)	**37.3**(46.55)	33(**2.08**)	32.7(9.97)	32.5(6.55)	32.8(5.26)	**39.3**(47.57)
scheduling (1)	1478(600.00)	1476(209.98)	1478(**29.94**)	1478(107.71)	**1481**(297.34)	1478(600.00)	1477(600.00)	1478(**29.94**)	1478(104.15)	**1481**(162.31)
SeanSafarpour (1)	NA(600.00)	NA(600.00)	**936004**(312.62)	**936004**(**44.42**)	NA(600.00)	NA(600.00)	NA(600.00)	**936004**(320.09)	**936004**(**48.05**)	NA(600.00)
treewidth-computation (3)	**52.8**(28.90)	**52.8**(26.56)	**52.8**(16.11)	**52.8**(**15.32**)	**52.8**(17.99)	**52.8**(37.75)	**52.8**(33.12)	**52.8**(16.09)	**52.8**(**15.25**)	**52.8**(20.22)
uaq (3)	140.3(10.06)	136.7(16.81)	137.3(25.72)	141.3(**7.88**)	**170**(79.88)	140.3(10.08)	138.3(17.95)	138.3(26.89)	149.3(**7.97**)	**173**(84.54)
xai-mindset2 (1)	374(0.07)	372(0.06)	374(**0.02**)	359(**0.02**)	**376**(22.43)	374(0.07)	372(0.07)	374(0.03)	360(**0.01**)	**376**(27.31)

solvers are executed once, and all other incomplete solvers, including LS-DTKMS, TOPKLS, and HEA-D are executed 10 times with different settings of random seeds. The cutoff time for them is set to 600 seconds. Note that we use the default values of all the parameters for TOPKLS, HEA-D, Open-WBO, Maxino, RC2-A, and MaxHS.

4.2 Experiment Results on Top-k MaxSAT

Table 2 and 3 show the results of the comparison of LS-DTKMS with four top-k MaxSAT based DTKMS solvers on 73 top-k MaxSAT instances (belong to 29 families). Since for some instances when k is set to 5 or more, all soft clauses are satisfied, we set the parameter k to 2, 3, 4, and 5, respectively. In the two tables, the first column records the name of each family as well as the number of instances in each family (in brackets). For each family of instances, we report the average number of variables ($|Var|$), hard clauses ($|Hard|$), and soft clauses ($|Soft|$) (Table 1), the average number of satisfied soft clauses and the average runtime in seconds (in brackets) that each solver can solve within the cutoff time (Table 2 and 3). If a solver fails to find a feasible solution, the corresponding result is marked with "NA". As shown in the two tables, LS-DTKMS achieves the best results in terms of solution quality on most families. In addition, as k gets larger, the solution quality of LS-DTKMS are almost stable. Hence, the performance of LS-DTKMS is considerably better than the ones of four top-k MaxSAT based DTKMS solvers. Moreover, the runtime of LS-DTKMS is comparable to the other solvers for the majority of instances.

Table 3 Comparison on top-k maxsat with k=4 and k=5.

	k=4					k=5				
Families (29)	Open-WBO	MaxHS	Maxino	RC2-A	LS-DTKMS	Open-WBO	MaxHS	Maxino	RC2-A	LS-DTKMS
aes (1)	NA(600.00)	132(25.53)	132(0.23)	132(**0.20**)	**147**(6.12)	NA(600.00)	132(25.95)	132(0.23)	132(**0.21**)	**147**(5.12)
aes-key-recovery (1)	NA(600.00)	406(600.00)	NA(600.00)	406(600.00)	**407**(**73.34**)	NA(600.00)	406(600.00)	NA(600.00)	406(600.00)	**407**(**75.12**)
atcoss (2)	265.5(108.59)	265.5(25.35)	265.5(22.44)	265.5(**53.06**)	**290.5**(57.96)	265.5(110.47)	265.5(26.79)	265.5(**22.90**)	265.5(57.42)	**290.5**(62.12)
bcp (2)	73.5(0.01)	73.5(0.01)	73(0.01)	73.5(**0.00**)	**74**(7.78)	73.5(**0.01**)	73.5(**0.01**)	73(**0.01**)	73.5(**0.01**)	**74**(6.99)
CircuitDebugging-Problems (2)	432736 (321.45)	**432736.5** (299.95)	432736 (**5.33**)	432736 (9.11)	**432736.5** (23.08)	**432736.5** (322.19)	**432736.5** (300.17)	**432736.5** (**6.04**)	**432736.5** (10.00)	**432736.5** (24.57)
CircuitTraceCo-mpaction (2)	13(20.56)	13(26.65)	13(**6.31**)	13(26.10)	**18.5**(14.03)	13(21.41)	13(30.88)	13(**6.25**)	13(26.68)	**20**(14.28)
close_solutions (4)	67858.3 (45.49)	67858.3 (87.84)	67858.3 (17.21)	67858.3 (**4.35**)	**67866** (37.92)	67858.3 (44.75)	67858.3 (130.86)	67858.3 (17.03)	67858.3 (**4.81**)	**67866** (40.90)
ConsistentQuery-Answering (3)	**0**(1.63)	**0**(0.05)	**0**(0.24)	**0**(**0.04**)	**0**(21.38)	**0**(1.65)	**0**(**0.05**)	**0**(0.25)	**0**(**0.05**)	**0**(22.46)
des (3)	3978.5(246.31)	3978.5(329.68)	4700(122.37)	4700.7(**31.35**)	**4701.7**(58.56)	4706.5(244.65)	4706.5(213.85)	4625.3(110.95)	4626(**55.35**)	**4626.3**(113.76)
drmx-atmostk (3)	31(0.67)	31.7(99.74)	22(201.60)	31.3(**21.58**)	**46.7**(116.70)	31(0.66)	31.7(96.77)	22(201.63)	31.7(**21.65**)	**46.7**(116.83)
fault-diagnosis (1)	**49593** (30.24)	**49593** (150.27)	**49593** (20.74)	**49593** (**7.59**)	**49593** (11.28)	**49593** (30.27)	**49593** (180.01)	**49593** (20.25)	**49593** (**7.69**)	**49593** (8.12)
frb (1)	45(43.69)	NA(600.00)	NA(600.00)	45(**7.00**)	**110**(99.09)	45(44.15)	NA(600.00)	NA(600.00)	45(**7.04**)	**122**(123.14)
gen-hyper-tw (1)	**44**(178.53)	**44**(**105.23**)	**44**(221.92)	**44**(116.71)	38(88.79)	**44**(181.03)	**44**(188.43)	**44**(223.14)	**44**(112.24)	42(**94.13**)
maxclique (3)	117(2.03)	116.7(60.98)	117(5.03)	117(**0.01**)	**143**(11.09)	117(1.50)	116.7(138.22)	117(5.00)	117(**0.00**)	**150.3**(10.49)
MaximumCom-monSub-Graph-Extraction (3)	23(219.66)	22.3(187.72)	23(107.63)	23(**104.37**)	**26**(151.78)	23(224.80)	22.3(197.61)	23(107.63)	23.7(**105.67**)	**27**(149.65)
maxone (2)	245(0.23)	243(0.18)	245(**0.05**)	238(0.22)	**258**(69.85)	248(0.17)	245(0.19)	245.5(**0.03**)	238(0.23)	**258**(64.77)
MaxSATQuerie-sinInterpretable-Classifiers (5)	528.2(22.02)	554.5(120.58)	526.6(**1.84**)	528.2(6.87)	**557.2**(67.97)	528.2(23.08)	554.5(110.08)	526.6(**1.87**)	528.2(6.93)	**557.2**(64.76)
mbd (2)	4617(259.82)	4617(42.93)	4616.5(1.85)	4617.5(0.69)	**4636**(**0.29**)	4617.5(255.90)	4617(47.07)	4617(1.83)	4618(0.67)	**4636**(**0.16**)
packup (5)	6972.4(11.34)	6972.4(7.04)	6973.2(0.99)	6971.2(**0.89**)	**7209.8**(28.74)	6972.6(11.38)	6972.6(7.32)	6973.4(0.98)	6973.2(**0.92**)	**7209.8**(26.52)
protein_ins (5)	31.6(91.45)	31.8(180.38)	31.6(209.63)	30.4(**34.42**)	**48**(104.57)	33.6(102.35)	33.6(174.08)	33.8(167.46)	31.4(**35.70**)	**55.4**(101.52)
pseudoBoolean (3)	408(0.82)	408.3(**0.05**)	408.3(0.07)	408(**0.05**)	**410**(1.96)	408(0.83)	408.3(**0.05**)	408.3(0.06)	408(**0.05**)	**410**(0.91)
railway-transport (2)	4777(21.63)	4777(38.16)	4777(7.38)	4777(**4.30**)	**4932.5**(23.28)	4777(27.77)	4777(38.07)	4777(7.28)	4777(**4.79**)	**4932.5**(24.23)
ramsey (1)	**210**(0.10)	**210**(0.70)	**210**(0.72)	**210**(**0.30**)	**210**(14.12)	**210**(**0.10**)	**210**(0.81)	**210**(0.73)	**210**(0.32)	**210**(10.31)
reversi (6)	33.8(**2.08**)	33.3(9.71)	33.3(6.92)	33.8(4.65)	**40.7**(48.64)	33.8(**2.21**)	33.3(19.36)	33.3(6.98)	33.8(5.12)	**41.3**(42.49)
scheduling (1)	**1481**(600.00)	1479(600.00)	1481(**37.57**)	1479(98.38)	**1481**(183.23)	1705(18.41)	1705(416.33)	1705(37.95)	1705(29.76)	**1707**(**18.01**)
SeanSafarpour (1)	NA(600.00)	NA(600.00)	**936004** (315.25)	**936004** (**45.29**)	NA(600.00)	NA(600.00)	NA(600.00)	**936004** (335.92)	**936004** (**56.57**)	NA(600.00)
treewidth-compu-tation (4)	**52.8**(41.45)	**52.8**(32.04)	**52.8**(14.79)	**52.8**(**14.38**)	**52.8**(18.57)	**52.8**(45.45)	**52.8**(31.81)	**52.8**(**14.75**)	**52.8**(15.61)	**52.8**(26.67)
uaq (3)	140.3(15.97)	138.3(20.87)	138.3(29.22)	155(**10.00**)	**183**(82.52)	140.3(16.46)	138.3(27.23)	138.3(31.06)	155(**11.57**)	**183**(73.48)
xai-mindset2 (1)	374(0.07)	372(0.08)	374(0.03)	361(**0.00**)	**376**(29.21)	374(0.07)	372(0.11)	374(**0.03**)	362(**0.03**)	**376**(45.13)

4.3 Experiment Results on DTKCS

In this subsection, we compare LS-DTKMS against two solvers on 37 DIMACS instances and set the parameter k to 5, 10, 15, and 20, respectively. To save space, we omit the experiment results that cannot be encoded into DTKMS and only display the ones of the 21 instances in Table 3 and 4. In the two tables, we report the number of hard clauses ($|hard|$), the number of soft clauses ($|soft|$), the largest number of satisfied soft clauses ($best$), the average one (avg), and the average time in seconds obtained by executing each solver ten times ($time$). Note that the scale of each instance is listed according to the transformed DIMACS instance using usual encoding, which causes the number of variables is equal to the number of soft clauses. The results show that the performance of LS-DTKMS is surprisingly good on most of instances. LS-DTKMS outperforms the other two solvers on all instances when k=15, and on most instances when k=5, 10 and 20. Specifically, LS-DTKMS cannot find the best solution for just 1 instance (C125.9 when k=10), which demonstrates that LS-DTKMS is a competitive DTKMS solver.

5 Related work

In this section, we review the related work, including the diversity in SAT and other diversified top-k problems.

Diversity in SAT. The diversity has been studied in the SAT problem and its related problems. For example, Agbaria et al. studied the diversity in SAT, whose diversity is measured by the value of the average distance between each pair of solutions normalized by the number of variables. They proposed two SAT-based methods to generate diverse

Table 4 Comparison on DTKCS with k=5 and k=10.

			k=5						k=10					
			TOPKLS		HEA-D		LS-DTKMS		TOPKLS		HEA-D		LS-DTKMS	
Instances (21)	|*Hard*|	|*Soft*|	*best*(*avg*)	*time*	*best*(*avg*)	*time*	*best*(*avg*)	*time*	*best*(*avg*)	*time*	*best*(*avg*)	*time*	*best*(*avg*)	*time*
brock200_2	10024	200	**52**(**52**)	110.63	**52**(**52**)	151.98	**52**(**52**)	53.96	93(93)	126.26	91(90)	210.78	**99**(**99**)	211.32
brock200_4	6811	200	73(73)	378.36	75(74)	169.93	**77**(**75**)	128.59	121(121)	36.53	133(130)	471.57	**135**(**133**)	35.17
brock400_2	20014	400	110(110)	187.87	115(112)	293.57	**120**(**120**)	207.95	192(192)	25.51	191(190)	263.77	**229**(**228**)	339.11
brock400_4	20035	400	119(119)	159.26	122(118)	352.24	**129**(**129**)	265	192(192)	530.14	191(190)	196.11	**231**(**231**)	225.13
C1000.9	49421	1000	271(270)	292.07	264(259)	534.38	**321**(**319**)	281.39	464(464)	497.64	474(471)	219.83	**507**(**507**)	116.46
C125.9	787	125	101(101)	73.04	113(112)	6.90	**123**(**122**)	5.89	104(104)	0.03	**125**(**125**)	0.00	122(120)	9.62
C250.9	3141	250	155(155)	553.33	173(167)	154.54	**193**(**193**)	28.48	206(206)	406.89	**250**(**249**)	0.02	**250**(246)	2.32
C500.9	12418	500	212(212)	222.91	224(219)	372.79	**257**(**256**)	147.64	323(321)	285.76	372(369)	329.21	**443**(**441**)	284.39
gen200_p0.9_44	1990	200	130(130)	561.09	149(145)	55.35	**174**(**174**)	223.79	166(166)	249.74	**200**(**200**)	0.00	**200**(197)	15.34
gen200_p0.9_55	1990	200	145(145)	392.94	160(157)	5.93	**185**(**185**)	106.08	174(174)	84.37	**200**(**200**)	0.00	**200**(**200**)	70.21
gen400_p0.9_55	7980	400	184(184)	193.76	200(197)	296.86	**219**(**207**)	209.95	264(264)	127.53	334(329)	179.58	**380**(**380**)	351.47
gen400_p0.9_65	7980	400	189(189)	283.49	217(210)	283.67	**250**(**250**)	218.55	274(274)	453.39	340(338)	262.97	**377**(**377**)	191.76
gen400_p0.9_75	7980	400	196(196)	402.29	240(232)	192.26	**265**(**265**)	220.12	276(276)	74.51	344(342)	113.96	**367**(**367**)	57.04
hamming8-4	11776	256	**80**(**80**)	10.05	**80**(**80**)	4.54	**80**(**80**)	8.33	**160**(**160**)	0.98	**160**(**160**)	75.85	**160**(**160**)	8.2
keller4	5100	171	**55**(**55**)	7.29	**55**(**55**)	11.01	**55**(**55**)	2.24	99(99)	274.42	98(97)	248.39	**110**(**110**)	114.16
MANN_a27	702	378	366(366)	524.8	**378**(**378**)	0.00	**378**(375)	260.1	**378**(**378**)	100.54	**378**(**378**)	0.00	**378**(**378**)	52.15
MANN_a45	1980	1035	973(973)	462.08	**1035**(**1035**)	0.01	**1035**(1032)	35.85	1034(1034)	62.14	**1035**(**1035**)	0.01	**1035**(**1035**)	66.36
MANN_a81	6480	3321	3036(3032)	335.67	**3321**(**3321**)	0.12	**3321**(3320)	31.47	3301(3301)	88.8	**3321**(**3321**)	0.12	**3321**(**3321**)	1.52
p_hat300-1	33917	300	**39**(**39**)	7.67	**39**(**39**)	55.04	**39**(**39**)	7.05	73(**73**)	85.72	68(68)	226.72	**74**(**73**)	268.54
p_hat300-2	22922	300	79(79)	222.49	80(77)	111.64	**94**(**93**)	55.15	110(110)	73.2	130(127)	373.91	**150**(**150**)	343.28
p_hat300-3	11460	300	108(108)	9.00	116(113)	232.32	**138**(**138**)	6.69	148(147)	251.25	193(187)	408.68	**195**(**195**)	109.7

solutions (satisfying assignments) of SAT for use in the hardware semiformal verification [1]. As a detailed extension of [1], Nadel further discussed the diverse$k$Set problem in SAT, that is, the problem of efficiently generating a number of diverse solutions given a formula [28]. Alòs et al. proposed a minimum decision tree computation algorithm based on MaxSAT encoding, where one of the tasks is to generate diverse solutions with different variable assignments to extract multiple minimum decision trees. The diversity is enforced by target variables during the incremental calls to the SAT solver, allowing the algorithm to favour the polarity of target variables that were less frequent in previous solutions [2].

Diversified top-k problem. The diversified top-k problem has been extensively studied, which aims to find diversified top-k results. [16, 32, 34, 37] studied the diversified top-k clique problem and [33] worked on the diversified top-k s-plex problem, both of which are to find k cliques or s-plex to maximize the number of covered vertices. Fan et al. studied the diversified top-k graph pattern matching problem, which to find a set of k matches such that the bi-criteria diversification function is maximized [14]. Liu et al. defined the k shortest paths with diversity problem of finding top-k shortest paths to minimizes the total length [24]. Xu et al. proposed two exact algorithms for the spatial diversified top-k routes (SDkR) query to obtain k trip routes with high popularity [36]. Lin et al. introduced the diversified top-k lasting cohesive subgraphs problem, which finds k maximal lasting (k, σ)-cores with maximum coverage regarding the number of vertices and timestamps [23]. Lyu et al. presented an algorithm for the diversified top-k biclique search problem which aims to find k maximal bicliques that cover the maximum number of edges [25]. Overall, the diversity top-k problem is becoming increasingly important research field.

6 Conclusion and Future Work

In this study, we propose a local search algorithm for DTKMS, called LS-DTKMS, which features scoring functions to select variables and assignments. The results show that LS-DTKMS achieves good performance across a broad range of instances, including the top-k MaxSAT instances and DTKCS instances. In the future, we will attempt to further improve LS-DTKMS via a few novel heuristic rules.

Table 5 Comparison on DTKCS with k=15 and k=20.

			k=15						k=20					
			TOPKLS		HEA-D		LS-DTKMS		TOPKLS		HEA-D		LS-DTKMS	
Instances (21)	$\|Hard\|$	$\|Soft\|$	*best(avg)*	*time*	*best(avg)*	*time*	*best(avg)*	*time*	*best(avg)*	*time*	*best(avg)*	*time*	*best(avg)*	*time*
brock200_2	10024	200	125(125)	296.52	128(126)	230.60	**143**(**143**)	70.22	146(146)	296.52	155(154)	372.79	**164**(**160**)	339.04
brock200_4	6811	200	151(151)	20.98	167(163)	410.85	**191**(**190**)	25.73	170(170)	20.98	192(188)	94.76	**198**(**196**)	44.24
brock400_2	20014	400	248(248)	491.45	264(263)	231.79	**320**(**320**)	220.73	292(292)	491.45	319(318)	240.85	**369**(**369**)	259.81
brock400_4	20035	400	252(252)	142.27	263(262)	305.84	**327**(**327**)	256.96	287(287)	142.27	325(318)	184.80	**360**(**360**)	180.30
C1000.9	49421	1000	587(587)	177.08	664(657)	136.11	**770**(**770**)	316.18	689(689)	177.08	841(833)	56.72	**823**(**823**)	372.70
C125.9	787	125	**125**(**125**)	0.00	**125**(**125**)	0.00	**125**(**125**)	19.74	**125**(**125**)	0.00	**125**(**125**)	0.00	**125**(**125**)	11.92
C250.9	3141	250	226(226)	86.74	**250**(**250**)	0.00	**250**(**250**)	119.19	223(223)	86.74	**250**(**250**)	0.00	**250**(**250**)	4.16
C500.9	12418	500	383(383)	57.54	490(488)	27.51	**500**(**500**)	142.67	418(418)	57.54	**500**(**500**)	0.00	**500**(**500**)	64.40
gen200_p0.9_44	1990	200	168(168)	2.83	**200**(**200**)	0.00	**200**(**200**)	9.16	**200**(**200**)	2.83	**200**(**200**)	0.00	**200**(**200**)	18.28
gen200_p0.9_55	1990	200	172(172)	1.61	**200**(**200**)	0.00	**200**(**200**)	11.77	**200**(**200**)	1.61	**200**(**200**)	0.00	**200**(**200**)	32.98
gen400_p0.9_55	7980	400	310(310)	43.22	**400**(**400**)	0.03	**400**(**400**)	9.42	326(325)	43.22	**400**(**400**)	0.01	**400**(**400**)	44.03
gen400_p0.9_65	7980	400	319(319)	88.31	**400**(**400**)	0.02	**400**(**400**)	12.81	341(341)	88.31	**400**(**400**)	0.01	**400**(**400**)	39.76
gen400_p0.9_75	7980	400	311(311)	442.95	**400**(**400**)	0.01	**400**(**400**)	11.35	335(335)	442.95	**400**(**400**)	0.00	**400**(**400**)	69.26
hamming8-4	11776	256	224(224)	69.23	**240**(229)	416.12	**240**(**240**)	41.98	246(246)	69.23	222(220)	353.77	**251**(**251**)	318.61
keller4	5100	171	126(126)	535.79	131(130)	218.73	**149**(**149**)	206.85	142(142)	535.79	152(**151**)	171.54	**153**(**151**)	52.11
MANN_a27	702	378	**378**(**378**)	0.14	**378**(**378**)	0.00	**378**(**378**)	6.13	**378**(**378**)	0.00	**378**(**378**)	0.00	**378**(**378**)	4.64
MANN_a45	1980	1035	**1035**(1033)	1.55	**1035**(**1035**)	0.01	**1035**(**1035**)	13.31	**1035**(**1035**)	1.55	**1035**(**1035**)	0.01	**1035**(**1035**)	57.91
MANN_a81	6480	3321	**3321**(**3321**)	338.63	**3321**(**3321**)	0.13	**3321**(**3321**)	116.70	**3321**(**3321**)	338.63	**3321**(**3321**)	0.13	**3321**(**3321**)	44.86
p_hat300-1	33917	300	**108**(105)	360.05	98(97)	198.69	**108**(**107**)	251.24	125(125)	360.05	121(118)	229.45	**140**(**140**)	224.28
p_hat300-2	22922	300	134(134)	57.61	162(159)	344.25	**191**(**191**)	257.23	144(144)	57.61	187(185)	194.29	**220**(**219**)	44.66
p_hat300-3	11460	300	168(168)	54.77	240(234)	156.88	**276**(**276**)	233.52	177(177)	54.77	275(**274**)	207.40	**276**(272)	46.18

References

1 Sabih Agbaria, Dan Carmi, Orly Cohen, Dmitry Korchemny, Michael Lifshits, and Alexander Nadel. Sat-based semiformal verification of hardware. In Roderick Bloem and Natasha Sharygina, editors, *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23*, pages 25–32. IEEE, 2010. doi:10.5555/1998496.1998505.

2 Josep Alòs, Carlos Ansótegui, and Eduard Torres. Interpretable decision trees through MaxSAT. *Artificial Intelligence Review*, pages 1–21, 2022. doi:10.1007/s10462-022-10377-0.

3 Mario Alviano. Maxino. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 21–22, 2020.

4 Josep Argelich, Chu Min Li, Felip Manyà, and Zhu Zhu. MinSAT versus MaxSAT for optimization problems. In Christian Schulte, editor, *Principles and Practice of Constraint Programming – 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 133–142. Springer, 2013. doi:10.1007/978-3-642-40627-0_13.

5 Giorgio Ausiello, Nicolas Boria, Aristotelis Giannakos, Giorgio Lucarelli, and Vangelis Th. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14):1901–1913, 2012. doi:10.1016/j.dam.2012.04.005.

6 Fahiem Bacchus. MaxHS in the 2020 MaxSAT evaluation. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 19–20, 2020.

7 Hamid Ahmadi Beni and Asgarali Bouyer. TI-SC: top-k influential nodes selection based on community detection and scoring criteria in social networks. *Journal of Ambient Intelligence and Humanized Computing*, 11(11):4889–4908, 2020. doi:10.1007/s12652-020-01760-2.

8 Jeremias Berg and Matti Järvisalo. Optimal correlation clustering via MaxSAT. In Wei Ding, Takashi Washio, Hui Xiong, George Karypis, Bhavani Thuraisingham, Diane J. Cook, and Xindong Wu, editors, *13th IEEE International Conference on Data Mining Workshops, ICDM Workshops, TX, USA, December 7-10, 2013*, pages 750–757. IEEE Computer Society, 2013. doi:10.1109/ICDMW.2013.99.

9 Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, António Morgado, and João Marques-Silva. Propositional proof systems based on maximum satisfiability. *Artificial Intelligence*, 300:103552, 2021. doi:10.1016/j.artint.2021.103552.

10 Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 747–753. AAAI Press, 2015. URL: `http://ijcai.org/Abstract/15/111`.

11 Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020. `doi:10.1016/j.artint.2020.103354`.

12 Mohamed Sami Cherif, Djamal Habet, and André Abramé. Understanding the power of max-sat resolution through up-resilience. *Artificial Intelligence*, 289:103397, 2020. `doi:10.1016/j.artint.2020.103397`.

13 Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. Efficient sampling of SAT solutions for testing. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 – June 03, 2018*, pages 549–559. ACM, 2018. `doi:10.1145/3180155.3180248`.

14 Wenfei Fan, Xin Wang, and Yinghui Wu. Diversified top-k graph pattern matching. *Proceedings of the VLDB Endowment*, 6(13):1510–1521, 2013. `doi:10.14778/2536258.2536263`.

15 Kevin Gimpel, Dhruv Batra, Chris Dyer, and Gregory Shakhnarovich. A systematic exploration of diversity in machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1100–1111. ACL, 2013. URL: `https://aclanthology.org/D13-1111/`.

16 Fei Hao, Zheng Pei, and Laurence T. Yang. Diversified top-k maximal clique detection in Social Internet of Things. *Future Generation Computer Systems*, 107:408–417, 2020. `doi:10.1016/j.future.2020.02.023`.

17 Hao Hu, Marie-José Huguet, and Mohamed Siala. Optimizing binary decision diagrams with MaxSAT for classification. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*, pages 3767–3775. AAAI Press, 2022. `doi:10.1609/aaai.v36i4.20291`.

18 Alexey Ignatiev. RC2-2018@ MaxSAT evaluation 2020. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 13–14, 2020.

19 Alexey Ignatiev, Yacine Izza, Peter J. Stuckey, and João Marques-Silva. Using MaxSAT for efficient explanations of tree ensembles. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*, pages 3776–3785. AAAI Press, 2022. `doi:10.1609/aaai.v36i4.20292`.

20 Mohit Kumar, Samuel Kolb, Stefano Teso, and Luc De Raedt. Learning MAX-SAT from contextual examples for combinatorial optimisation. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 4493–4500. AAAI Press, 2020. `doi:10.1609/aaai.v34i04.5877`.

21 Zhendong Lei and Shaowei Cai. Solving (weighted) partial maxsat by dynamic local search for SAT. In Jérôme Lang, editor, *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1346–1352. ijcai.org, 2018. `doi:10.24963/ijcai.2018/187`.

22 Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In Maria Fox and David Poole, editors, *Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. `doi:10.1609/aaai.v24i1.7536`.

23 Longlong Lin, Pingpeng Yuan, Rong-Hua Li, and Hai Jin. Mining diversified top-r lasting cohesive subgraphs on temporal networks. *IEEE Transactions on Big Data*, 8(6):1537–1549, 2022. `doi:10.1109/TBDATA.2021.3058294`.

24 Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. Finding top-k shortest paths with diversity. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):488–502, 2018. doi:10.1109/TKDE.2017.2773492.

25 Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. Maximum and top-k diversified biclique search at scale. *The VLDB Journal*, 31(6):1365–1389, 2022. doi:10.1007/s00778-021-00681-6.

26 Ruben Martins, Norbert Manthey, Miguel Terra-Neves, Vasco Manquinho, and Inês Lynce. Open-WBO@ MaxSAT evaluation 2020. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 24–25, 2020.

27 Christian J. Muise, Sheila A. McIlraith, and J. Christopher Beck. Optimally relaxing partial-order plans with MaxSAT. In Lee McCluskey, Brian Charles Williams, José Reinaldo Silva, and Blai Bonet, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI, 2012. doi:10.1609/icaps.v22i1.13537.

28 Alexander Nadel. Generating diverse solutions in SAT. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing – SAT 2011 – 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2011. doi:10.1007/978-3-642-21581-0_23.

29 Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. An efficient algorithm for computing the distance between close partitions. *Discrete Applied Mathematics*, 159(1):53–59, 2011. doi:10.1016/j.dam.2010.09.002.

30 Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. Spacing memetic algorithms. In Natalio Krasnogor and Pier Luca Lanzi, editors, *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 1061–1068. ACM, 2011. doi:10.1145/2001576.2001720.

31 Caleb Voss, Mark Moll, and Lydia E. Kavraki. A heuristic approach to finding diverse short paths. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 4173–4179. IEEE, 2015. doi:10.1109/ICRA.2015.7139774.

32 Jun Wu, Chu-Min Li, Lu Jiang, Junping Zhou, and Minghao Yin. Local search for diversified top-k clique search problem. *Computers & Operations Research*, 116:104867, 2020. doi:10.1016/j.cor.2019.104867.

33 Jun Wu, Chu-Min Li, Luzhi Wang, Shuli Hu, Peng Zhao, and Minghao Yin. On solving simplified diversified top-k s-plex problem. *Computers & Operations Research*, 153:106187, 2023. doi:10.1016/j.cor.2023.106187.

34 Jun Wu, Chu-Min Li, Yupeng Zhou, Minghao Yin, Xin Xu, and Dangdang Niu. HEA-D: A hybrid evolutionary algorithm for diversified top-k weight clique search problem. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 4821–4827. International Joint Conferences on Artificial Intelligence Organization, July 2022. doi:10.24963/ijcai.2022/668.

35 Mingyu Xiao. An exact MaxSAT algorithm: Further observations and further improvements. In Luc De Raedt, editor, *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 1887–1893. ijcai.org, 2022. doi:10.24963/ijcai.2022/262.

36 Hongfei Xu, Yu Gu, Jianzhong Qi, Jiayuan He, and Ge Yu. Diversifying top-k routes with spatial constraints. *Journal of Computer Science and Technology*, 34(4):818–838, 2019. doi:10.1007/s11390-019-1944-6.

37 Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Diversified top-k clique search. *The VLDB Journal*, 25(2):171–196, 2016. doi:10.1007/s00778-015-0408-z.

38 Xiaoqi Zheng, Taigang Liu, Zhongnan Yang, and Jun Wang. Large cliques in arabidopsis gene coexpression network and motif discovery. *Journal of plant physiology*, 168(6):611–618, 2011. doi:10.1016/j.jplph.2010.09.010.

A Comparison of SAT Encodings for Acyclicity of Directed Graphs

Neng-Fa Zhou
The City University of New York, NY, USA
Relational AI, Berkeley, CA, USA

Ruiwei Wang
National University of Singapore, Singapore

Roland H. C. Yap
National University of Singapore, Singapore

Abstract

Many practical applications require synthesizing directed graphs that satisfy the acyclic constraint along with some side constraints. Several methods have been devised for encoding acyclicity of directed graphs into SAT, each of which is based on a cycle-detecting algorithm. The leaf-elimination encoding (LEE) repeatedly eliminates leaves from the graph, and judges the graph to be acyclic if the graph becomes empty at a certain time. The vertex-elimination encoding (VEE) exploits the property that the cyclicity of the resulting graph produced by the vertex-elimination operation entails the cyclicity of the original graph. While VEE is significantly smaller than the transitive-closure encoding for sparse graphs, it generates prohibitively large encodings for large dense graphs. This paper reports on a comparison study of four SAT encodings for acyclicity of directed graphs, namely, LEE using unary encoding for time variables (LEE-u), LEE using binary encoding for time variables (LEE-b), VEE, and a hybrid encoding which combines LEE-b and VEE. The results show that the hybrid encoding significantly outperforms the others.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning; Computing methodologies → Planning and scheduling; Hardware → Theorem proving and SAT solving

Keywords and phrases Graph constraints, Acyclic constraint, SAT encoding, Graph Synthesis

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.30

Funding R. Wang and R. H. C. Yap's work was partly suported by NUS grant T1 251RES2219.

1 Introduction

Many practical combinatorial problems require synthesizing acyclic directed graphs, such as utility networks, planning problems [9], Markov networks [2], neural networks, and Bayesian networks [1, 8]. As there are side constraints involved beyond acyclic constraints, these problems are more complicated than simply checking if a synthesized subgraph is acyclic. In this paper, we focus on how to solve such problems with SAT solvers by encoding the acyclic constraint into SAT. While efficient algorithms exist for checking the acyclicity of directed graphs, it is still unknown which method performs well for encoding acyclicity of directed graphs into SAT.

Several methods have been devised for encoding acyclicity of directed graphs into SAT, each of which is based on a cycle-detecting algorithm. The leaf-elimination encoding (LEE), which is basically the same as the tree-reduction encoding [4] and the binary labeling encoding [6], is inspired by the leaf-elimination algorithm. The algorithm repeatedly eliminates leaves from the graph, and judges the graph to be acyclic if the graph becomes empty at a certain time.

26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023).
Editors: Meena Mahajan and Friedrich Slivovsky; Article No. 30; pp. 30:1–30:9
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Another approach is the transitive-closure encoding for directed graphs, which relies on the proposition that a graph's transitive closure preserves the graph's acyclicity. However the transitive closure encoding can be prohibitively large. For that reason, GraphSAT, which is an SMT solver integrating acyclicity checking into SAT solving, has been proposed [4]. Recently, the vertex-elimination encoding (VEE) [10] was proposed, which exploits the property that the cyclicity of the resulting graph produced by the vertex-elimination operation [11] entails the cyclicity of the original graph. VEE can be significantly smaller than the transitive-closure encoding for sparse graphs. However, for dense graphs, it is asymptotically the same as the transitive-closure encoding.

This paper first presents a comparison study of three SAT encodings for acyclicity of directed graphs, namely, LEE using unary encoding for time variables (LEE-u), LEE using binary encoding for time variables (LEE-b), and VEE. This paper then proposes a hybrid encoding, which combines the strengths of LEE-b and VEE. The hybrid encoding starts with VEE, and switches to LEE-b when the resulting graph after vertex elimination becomes dense. The experimental results show that the hybrid encoding significantly outperforms the others.

2 Preliminaries

The constraint $acyclic\_d(V, E)$ takes a base directed graph $G = (V, E)$, where V is a set of vertices, and E is a set of directed edges. The task is to synthesize a acyclic subgraph of G. More precisely, each vertex in V has a binary variable, called a *characteristic variable*, associated with it, which is 1 iff the vertex is in the subgraph to be synthesized. Each edge in E also has an associated characteristic variable, which indicates if the edge is in the subgraph. The constraint $acyclic\_d(V, E)$ is true if the subgraph of G determined by the characteristic variables is acyclic.

In the following, we use the notation (u, v) to denote a directed edge from vertex u to vertex v. The function $b(x)$ returns the characteristic variable of x. A vertex v is called an *in-vertex* if $b(v) = 1$, and an edge $e = (v_1, v_2)$ is called an *in-edge* if $b(e) = 1$. For each edge $e = (v_1, v_2)$ in E, $b(e) \to b(v_1) \wedge b(v_2)$, meaning that if an edge is in the subgraph, then both end vertices connected by the edge must be in the subgraph as well.

For a directed graph $G = (V, E)$, the function $nbs^-(v)$ returns the set of *in-neighbors* connected to v by incoming edges in the base graph:

$$nbs^-(v) = \{u \mid (u, v) \in E\}$$

and the function $nbs^+(v)$ returns the set of *out-neighbors* connected to v by outgoing edges in the base graph:

$$nbs^+(v) = \{u \mid (v, u) \in E\}.$$

A vertex v is said to be *peripheral* if either it has no in-neighbors or it has no out-neighbors. In particular, a peripheral vertex v is called a *leaf* in this paper if it has no out-neighbors. Note that singleton vertices that are not connected to any other vertices are treated as leaves.

3 The Leaf-Elimination Encoding (LEE)

Given a directed graph, the leaf-elimination algorithm detects cycles by repeatedly eliminating leaves from the graph from time 0 to a certain maximum time. If the graph is empty at the maximum time, then the graph is acyclic; otherwise, the graph is cyclic.

The maximum time is determined by the longest path in the graph. As the graph that comprises a list of linearly connected vertices has the longest path, the maximum time is upper bounded by n, where $n = |V|$. While finding the longest path in a graph is NP-hard, a tight upper bound can be obtained in some cases.

There is a straightforward CSP (Constraint Satisfaction Problem) model for the acyclicity constraint, which mimics the leaf-elimination algorithm for detecting cycles. For each vertex, a variable, called a *time variable*, is utilized to indicate the time at which the vertex becomes a leaf and is removed from the graph. The domain of the time variables is $0..m$, where m is the maximum time. The constraints ensure that only leaves can be removed at each time, and the graph must be empty at time m. If the graph is cyclic, then the CSP model is unsatisfiable. The encodings of the CSP model into SAT are called *leaf-elimination encodings* (LEE). Different methods can be utilized to encode time variables, such as unary encoding (also called direct encoding [3]) and binary encoding (also called log encoding [5]). When binary encoding is used for time variables, LEE, called LEE-b, is compact and can encode large graphs. However, it is well known that binary encoding has weak propagation strength [7], yet, the results in Section 6 show that the binary encoding pays off.

3.1 LEE-u

LEE-u (similar to tree reduction in [10]) employs a matrix of binary variables A of size $n \times m$, where $n = |V|$, the number of vertices in the base graph, and m is the maximum time. The entry $A_{v,t}$ is 1 if and only if vertex v has been eliminated by time t. The encoding imposes the following constraints on the variables:

$$\text{For } v \in V\text{: } \textstyle\sum_{u \in nbs^{+}(v)} b((v,u)) = 0 \leftrightarrow A_{v,0} \tag{1}$$

$$\text{For } v \in V,\ t \in 1..m\text{: } A_{v,t-1} \rightarrow A_{v,t} \tag{2}$$

$$\text{For } v \in V,\ t \in 1..m,\ u \in nbs^{+}(v)\text{: } b((v,u)) \wedge \neg A_{u,t-1} \rightarrow \neg A_{v,t} \tag{3}$$

$$\text{For } v \in V\text{: } A_{v,m} \tag{4}$$

Constraint (1) states that all leaves, i.e., vertices that have no outgoing edges, are eliminated at time 0. Constraint (2) enforces that once a vertex is eliminated, it is eliminated forever. Constraint (3) entails that a vertex cannot be eliminated at time t if any of its out-neighbors is not eliminated at time $t-1$. Constraint (4) forces every vertex to be eliminated at time m.

The correctness of the encoding is supported by the fact that the vertices that occur in a cycle can never be eliminated, i.e. $A_{v,t}$ cannot be equal to 1 for $t \in 1 \ldots m$, and constraints (3) and (4) are unsatisfiable for any vertex in a cycle.

The number of variables, besides the characteristic variables, used by LEE-u is the size of A, which is $O(n \times m)$. The number of clauses used by LEE-u, which is dominated by Constraint (3), is $O(n \times m \times d)$, where d is the average degree of the vertices in the base graph.

3.2 LEE-b

LEE-b is a variant of LEE, which encodes time variables using binary encoding. Binary encoding is more compact than unary encoding. It employs a sequence of binary variables for encoding a domain variable. Each combination of values of the binary variables represents a value for the domain variable. If there are holes in the domain, then not-equal constraints are generated to disallow assigning those hole values to the variable. Also, inequality constraints ($\geq$ and $\leq$) are generated to prohibit assigning out-of-bounds values to the variable if either bound is not $2^k - 1$ for some k. A detailed description of general techniques for binary encodings for domain variables and primitive constraints can be found in [13].

LEE-b uses a variable T_v with the domain $0..m$ for each vertex v in V, where m is the maximum time. LEE-b imposes the following constraints on time variables:

$$\text{For } v \in V\text{: } \sum_{u \in nbs^+(v)} b((v,u)) = 0 \leftrightarrow T_v = 0 \tag{5}$$

$$\text{For } v \in V,\ u \in nbs^+(v)\text{: } b((v,u)) \rightarrow T_v > T_u \tag{6}$$

Constraint (5) states that $T_v = 0$ if and only if vertex v is a leaf. Constraint (6) entails that for each directed edge (v,u), $T_v > T_u$, meaning that vertex v is removed after vertex u. The encoding of $T_v > T_u$ can be found in [12].

The correctness of the encoding is supported by the fact that the constraint $T_v > T_u$ is not commutative and constraint (6) is unsatisfiable if u and v occur in a cycle.

The number of variables, besides the characteristic variables, used by LEE-b is $O(n \times log_2(m))$. The number of clauses used by LEE-b is $O(n \times log_2(m) \times d)$, where d is the average degree of the vertices in the base graph.

4 Vertex-Elimination Encoding

The vertex-elimination encoding (VEE) [10] exploits the fact that the sequence of graphs produced by the vertex elimination operation [11] with respect to an elimination ordering preserves the acyclicity of the original graph.

Let O be an elimination ordering, $O = [v_1, v_2, \ldots, v_n]$, and G_0 be the original directed graph, $G_0 = ([v_1, v_2, \ldots, v_n], E_0)$. It is assumed that there are no vertices with self-loops in G_0. VEE produces a sequence of graphs G_1, G_2, $\ldots$, G_n by eliminating vertices according to the elimination ordering. The *vertex-elimination graph* is the union of the graphs $G^* = G_0 \bigcup G_1 \ldots \bigcup G_n$. Let $G_{i-1} = ([v_i, \ldots, v_n], E_{i-1})$. The graph $G_i = ([v_{i+1}, \ldots, v_n], E_i)$ is obtained by eliminating v_i from G_{i-1}, where

$$\begin{aligned} E_i = E_{i-1}\ - & \\ & \{(u, v_i) | u \in nbs^-(v_i)\}\ - \\ & \{(v_i, u) | u \in nbs^+(v_i)\}\ + \\ & \{(u, w) | u \in nbs^-(v_i),\ \ w \in nbs^+(v_i), u \neq w\}. \end{aligned}$$

The operation eliminates v_i's adjacent edges, and adds the edge (u,w) into E_i for each u in v_i's in-neighbors and each w in v_i's out-neighbors if $u \neq w$ and the edge is not contained in E_i. Each newly added edge (u,w) is attached with a characteristic binary variable $b((u,w))$. In addition, for all u in $nbs^-(v_i)$ and w in $nbs^+(v_i)$ such that $u \neq w$, the variable $b((u,w))$ is entailed by the variables $b((u,v_i))$ and $b((v_i,w))$:

$$b((u, v_i))\ \wedge\ b((v_i, w))\ \rightarrow\ b((u, w)). \tag{7}$$

For each u in $nbs^-(v_i)$, if $u \in nbs^+(v_i)$, VEE, after eliminating v_i, generates the following constraint to ensure that there is no cycle between u and v_i in E_{i-1}

$$\neg b((u, v_i))\ \vee\ \neg b((v_i, u)). \tag{8}$$

Constraints (7) and (8) ensure that the acyclicity of E_i entails the acyclicity of E_{i-1}. Therefore, with the accumulated constraints, E_0 is acyclic if E_i is acyclic by induction on i.

The correctness of VEE is guaranteed by the fact that a cycle of any length in G_0 will lead to a cycle of size 2 in G_{i-1} ($i \in 1..n$), which will make constraint (8) unsatisfiable.

The number of binary variables used by VEE, which is upper bounded by $O(n^2)$, is determined by the number of edges in the original graph and the number of new edges added during the vertex elimination process. The number of clauses generated by VEE is $O(d^2 \times n)$, where $n =|V|$ and d is the average degree of the vertex-elimination graph. In comparison, the transitive closure encoding uses $O(n^2)$ variables and generates $O(n^3)$ clauses [10]. While VEE is significantly more compact than the transitive clause encoding for sparse graphs, it is asymptotically the same as the transitive-closure encoding in the worst case.

5 Hybrid Encoding

The resulting graph obtained after each vertex elimination tends to be more dense than the original graph, and VEE becomes closer to the transitive-closure encoding. Thus, the encoding may become prohibitively large. One idea to alleviate code explosion while harnessing the propagation strengths of VEE is to start with VEE when the graph is sparse, and switch to LEE-b when the graph becomes dense. We call this encoding that combines VEE and LEE-b a *hybrid encoding*.

Formally, given any elimination ordering $[v_1, v_2, \ldots, v_n]$ and a switch position $0 \leq i \leq n$, the hybrid encoding uses the constraints (7) and (8) of VEE to generate the graph G_i by eliminating the vertices $v_1, \cdots, v_i$ from the original graph G_0. Then constraints (5) and (6) of LEE-b are used to encode the acyclicity of G_i.

The correctness of a hybrid encoding is guaranteed by the correctness of VEE and LEE-b. For any $i \in 0 \ldots n$, constraints (7) and (8) ensure that if G_i is acyclic then G_0 is also acyclic. If LEE ensures that G_i is acyclic, and VEE ensures that there are no cycles in G_1, G_2, …, G_{i-1}, then the hybrid encoding also ensures that the graph G_0 is acyclic. The encoding size depends on the encoding sizes of VEE and LEE-b, as well as the switching heuristic.

A concrete hybrid encoding needs to decide when to switch to LEE-b. In our implementation, the hybrid encoding uses the *mindegree* heuristic (selecting a vertex with the smallest total number of incoming and outgoing edges in the graph generated by the vertex elimination). It switches to LEE-b if the current graph $G^c = G_0 \bigcup G_1 \cdots \bigcup G_i$ contains 2.3 times as many edges as the original graph G_0 or the current graph G^c contains more than $30 \times n$ edges based on preliminary experiments, where n is the number of vertices in the original graph G_0.

6 Experimental Results

All the encodings discussed above have been implemented in Picat[1] (version 3.4). Picat provides a state-of-art SAT-based CSP solver. For example, it won two gold medals in the 2022 XCSP solver competition[2] and two silver medals in the 2022 MiniZinc Challenge.[3] Picat encodes constraints into SAT and employs Kissat[4] as the underlying SAT solver.

This section presents the results of an experiment comparing the encodings on the GraphSAT benchmarks.[5] The benchmarks consist of five categories of instances with graph sizes ranging from 15 to 10002 vertices. The GraphSAT benchmarks are modelled with

[1] http://picat-lang.org

[2] https://www.xcsp.org/competitions/

[3] https://www.minizinc.org/challenge2022/results2022.html

[4] https://github.com/arminbiere/kissat

[5] https://users.aalto.fi/~rintanj1/software.html

Table 1 Summary of results.

#Insts	Benchmark	LEE-u	LEE-b	VEE	HYB	Virtual-HYB
26	COMB	28.03	42.54	**0.13**	**0.13**	**0.13**
31	EMPTYCORNER	160.89	1.99	3.84	1.99	**1.51**
36	EMPTYMIDDLE	67.01	6.93	0.92	2.02	**0.87**
9	ESCAPE	379.45	22.74	285.91	23.64	**15.38**
14	ROOMCHAIN	343.35	342.99	15.38	15.34	**11.94**
116	**TOTAL**	16350.64	6423.84	2944.2	565.29	**387.09**
	AVE	140.95	55.38	25.38	4.87	**3.34**
	#SOLVED	94	108	112	**116**	**116**

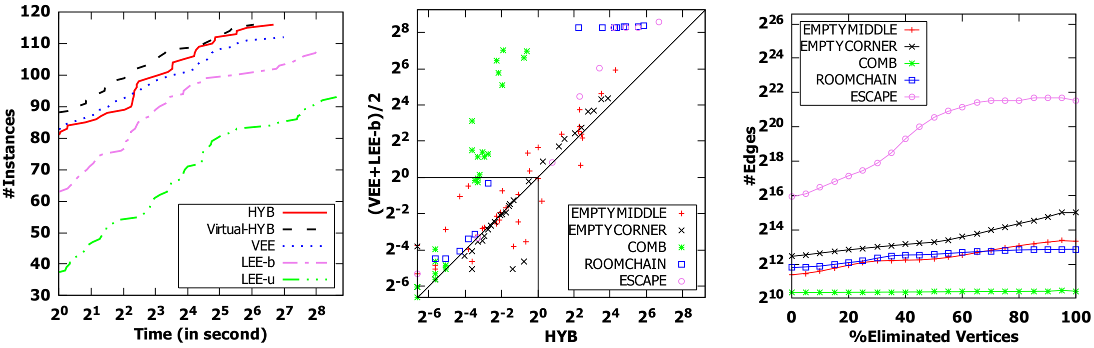
conjunctive-normal-form clauses and some graph constraints encoded as acyclic constraints. These benchmarks are introduced by [9] and have been also used to evaluate implementations of the acyclic constraint in [10]. All the CPU times reported below were measured on Linux Ubuntu with a 3.20GHz and 16G RAM Intel i7-8700 machine. The time limit used was 10 minutes per instance.

Table 1 gives a summary of the experimental results, where the column **#Insts** indicates the number of instances, the column **Benchmark** indicates the benchmark category, and each of the remaining columns indicates the average CPU time taken by an encoding. **Virtual-HYB** is the "virtual best" encoding among 21 different hybrid encodings, each of which uses VEE to eliminate $p \in \{0, 5, \cdots, 100\}$ percent of vertices and then switches to LEE-b. Note that the hybrid encoding with $p = 0$ is equivalent to **LEE-b**, and the hybrid encoding with $p = 100$ is **VEE**. The row **Total** gives the total CPU time for all instances, **AVE** the average CPU time per instance, and **#Solved** the number of solved instances within the time limit. In the experiments, the maximum time m used in **LEE** is n.[6]

It can be seen that, among the four encodings, **HYB** performs the best. **HYB** succeeds on all of the 116 instances, while **LEE-u**, **LEE-b**, and **VEE**, respectively, fail on 22, 8 and 4 instances. It can also be seen that while both **LEE-u** and **LEE-b** are based on the idea of leaf elimination, **LEE-b** is much better than **LEE-u**. On average, **HYB** is 5 times as fast as **VEE**, and 10 times as fast as **LEE-b**. **HYB** is also more robust than **VEE** and **LEE-b**. For each category, **HYB** has similar performance to the best of **LEE-u**, **LEE-b**, and **VEE**. **Virtual-HYB** performs the best on each category, but it is intended as a comparison with a form of virtual best heuristic. The results of **Virtual-HYB** entail that there is potential to improve the hybrid encoding by giving a better heuristic to decide when to switch from **VEE** to **LEE-b**.

Figure 1a gives the runtime distribution of the five encodings. **Virtual-HYB** overall outperforms the other encodings, and **HYB** always solves more instances than **VEE**, **LEE-b** and **LEE-u** when the solving time limit is set to more than 6 seconds. Also as the solving time increases, **HYB** gets close to **Virtual-HYB**. Figure 1b compares the solving time of **HYB** with $(\mathbf{VEE} + \mathbf{LEE\text{-}b})/2$, the average CPU time of **VEE** and **LEE-b**. Each dot in the figure denotes an instance. It can be seen that **HYB** is faster than the average of **VEE** and **LEE-b** on most non-trivial instances (i.e. the instances not solved in 1 second by either **VEE** or **LEE-b**). This illustrates that **HYB** encoding is robust. The performance of **HYB** can be closer to the best between **VEE** and **LEE-b**. Figure 1c shows the average number of

[6] We experimented with optimizing for some special cases, such as removing peripheral vertices, but the difference was negligible.



(a) Runtime distribution. **(b)** The robustness of HYB. **(c)** The average number of edges.

Figure 1 Detailed comparisons between the encodings.

Table 2 A comparison on encoding sizes.

Benchmark	|V|	|E|	LEE-u		LEE-b		VEE		HYB	
			#vars	#cls	#vars	#cls	#vars	#cls	#vars	#cls
COMB005-3	134	1215	24473	208373	13559	86741	4241	13483	4241	13483
COMB010-3	389	3595	170363	1599113	43319	276548	12306	39618	12306	39618
EMPTYCORNER024-1	578	2877	347359	1697K	37951	164478	18990	105021	35753	209184
EMPTYCORNER075-1	5252	26247	fail	fail	427631	1945K	255548	4491K	459127	2680K
EMPTYMIDDLE025-2	1252	10610	fail	fail	155233	878838	75075	883099	159303	1071K
EMPTYMIDDLE030-1	902	4497	834330	4110K	59490	260162	32174	224943	61454	359879
escape08-1	4098	89688	fail	fail	1269K	8047K	fail	fail	1618K	11180K
escape10-1	10002	229252	fail	fail	3675K	23235K	fail	fail	4569K	30856K
ROOMCHAIN005-2	510	4223	282256	2385K	52076	295302	19328	76752	19328	76752
ROOMCHAIN006-3	917	11025	901673	10932K	147290	866044	49244	253186	49244	253186

edges in the vertex-elimination graph of each category generated by using **VEE** to eliminate $p \in \{0, 5, \cdots, 100\}$ percent of vertices. We can see that **VEE** is much faster than **LEE-b** on the categories where the graph size is small and grows slowly, such as the COMB and ROOMCHAIN categories.

Table 2 gives the graph sizes of 10 selected instances and the encoding sizes. The column |**V**| gives the number of vertices, and |**E**| gives the number of edges in the base graph. For each encoding, the table gives the number of variables (#vars) and the number of clauses (#cls) in the generated code. The entry *fail* indicates that the encoder fails to finish generating the encoding, i.e. the encoder runs out of memory or time. **LEE-u** fails on 4 of the instances, **VEE** fails on 2 of the instances, while **LEE-u** and **HYB** succeed on all the instances. Naturally for failed encodings, the SAT solver is never invoked. The results also show that **LEE-u** produces the largest encodings, and for the EMPTYMIDDLE instances the encoder runs out of memory. This may explain why the performance of **LEE-u** is also the worst.

Table 3 gives the CPU time of each run, which includes both the translation time and solving time. The entry *T.O.* indicates that the run does not finish within the time limit. **HYB** succeeds in solving all the 10 instances, while each of the other encoders fails to solve some of the instances. In addition, **HYB** is faster than both **VEE** and **LEE-b** on some instances, such as the EMPTYCORNER075-1 instance. **LEE-b** is significantly faster than **VEE** on instances where **VEE** fails, such as escape08-01.

Table 3 A comparison on CPU times.

Benchmark	LEE-u	LEE-b	VEE	HYB
COMB005-3	7.21	17.36	**0 .08**	**0.08**
COMB010-3	115.17	195.78	**0.59**	**0.59**
EMPTYCORNER024-1	26.86	**0.3**	0.42	0.35
EMPTYCORNER075-1	fail	14.52	24.58	**11.5**
EMPTYMIDDLE025-2	fail	43.15	**5.66**	11.49
EMPTYMIDDLE030-1	26.44	0.84	1.05	**1.03**
escape08-1	fail	**15.96**	fail	29.04
escape10-1	fail	150.98	fail	**101.84**
ROOMCHAIN005-2	T.O.	T.O.	**45.55**	45.79
ROOMCHAIN006-3	T.O.	T.O.	**11.56**	11.61

7 Discussion and Conclusion

This paper compares several SAT encodings for the acyclic constraint on directed graphs. For LEE, this paper compares two encodings of time variables, and shows that the decision on which encoding to select has a great impact on the performance. While LEE-b is compact, it fails on some mid-sized instances due to its weak propagation caused by the binary representation of time variables. Our study also confirms that, while VEE is effective for sparse mid-sized graphs, it generates prohibitively large encodings for larger graphs even when the graphs are sparse.

Most importantly, our study finds that the hybrid encoding, which starts with VEE and switches to LEE-b when the graph becomes dense, significantly outperforms both LEE-b and VEE. The good performance of the hybrid encoding is attributed to its combination of VEE's strong propagation and LEE-b's conciseness. The hybrid encoding clearly advances the state of the art. Ultimately, to solve more difficult problems, scalability is needed. The hybrid encoding will be a new starting point for future researchers and practitioners.

As the idea of hybridization is new in this context, it warrants more investigation. For example, further work needs to be done to find even better heuristics for switching from VEE to LEE-b. Also, the idea of hybridization may also be effective for encoding other graph constraints, such as the reachability constraint.

References

1 Jeremias Berg, Matti Järvisalo, and Brandon Malone. Learning Optimal Bounded Treewidth Bayesian Networks via Maximum Satisfiability. In Samuel Kaski and Jukka Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 86–95, Reykjavik, Iceland, 22–25 April 2014. PMLR. URL: `https://proceedings.mlr.press/v33/berg14.html`.

2 Jukka Corander, Tomi Janhunen, Jussi Rintanen, Henrik J. Nyman, and Johan Pensar. Learning chordal markov networks by constraint satisfaction. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 1349–1357, 2013. URL: `https://proceedings.neurips.cc/paper/2013/hash/c06d06da9666a219db15cf575aff2824-Abstract.html`.

3 Johan de Kleer. A comparison of ATMS and CSP techniques. In N. S. Sridharan, editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, pages 290–296. Morgan Kaufmann, 1989. URL: `http://ijcai.org/Proceedings/89-1/Papers/046.pdf`.

4 Martin Gebser, Tomi Janhunen, and Jussi Rintanen. SAT modulo graphs: Acyclicity. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence – 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2014. doi:10.1007/978-3-319-11558-0_10.

5 Kazuo Iwama and Shuichi Miyazaki. Sat-varible complexity of hard combinatorial problems. In Björn Pehrson and Imre Simon, editors, *Technology and Foundations – Information Processing '94, Volume 1, Proceedings of the IFIP 13th World Computer Congress, Hamburg, Germany, 28 August – 2 September, 1994*, volume A-51 of *IFIP Transactions*, pages 253–258. North-Holland, 1994.

6 Mikolas Janota, Radu Grigore, and Vasco M. Manquinho. On the quest for an acyclic graph. *CoRR*, abs/1708.01745, 2017. arXiv:1708.01745.

7 Donald Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley, 2015.

8 Zhenyu A. Liao, Charupriya Sharma, James Cussens, and Peter van Beek. Finding all bayesian network structures within a factor of optimal. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 – February 1, 2019*, pages 7892–7899. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33017892.

9 Binda Pandey and Jussi Rintanen. Planning for partial observability by SAT and graph constraints. In Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs T. J. Spaan, editors, *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, pages 190–198. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17756.

10 Masood Feyzbakhsh Rankooh and Jussi Rintanen. Propositional encodings of acyclicity and reachability by using vertex elimination. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 – March 1, 2022*, pages 5861–5868. AAAI Press, 2022. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20530.

11 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.

12 Neng-Fa Zhou and Håkan Kjellerstrand. The Picat-SAT compiler. In Marco Gavanelli and John H. Reppy, editors, *Practical Aspects of Declarative Languages – 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings*, volume 9585 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2016. doi:10.1007/978-3-319-28228-2_4.

13 Neng-Fa Zhou and Håkan Kjellerstrand. Optimizing SAT encodings for arithmetic constraints. In *CP*, pages 671–686, 2017. doi:10.1007/978-3-319-66158-2_43.