Lecture Notes in Computer Science 4501

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

João Marques-Silva Karem A. Sakallah (Eds.)

Theory and Applications of Satisfiability Testing – SAT 2007

10th International Conference
Lisbon, Portugal, May 28-31, 2007
Proceedings

Springer

Volume Editors

João Marques-Silva
University of Southampton
School of Electronics and Computer Science
Highfield, Southampton, S017 1BJ, UK
E-mail: jpms@ecs.soton.ac.uk

Karem A. Sakallah
University of Michigan
Department of Electrical and Computer Science
4603 CSE Building, 2260 Hayward Ave, Ann Arbor, MI 48109-2121, USA
E-mail: karem@umich.edu

Library of Congress Control Number: 2007927094

CR Subject Classification (1998): F.4.1, I.2.3, I.2.8, I.2, F.2.2, G.1.6

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

ISSN 0302-9743
ISBN-10 3-540-72787-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-72787-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12069392 06/3180 5 4 3 2 1 0

Preface

This volume contains the papers presented at SAT 2007: 10th International Conference on Theory and Applications of Satisfiability Testing.

The International Conferences on Theory and Applications of Satisfiability Testing (SAT) originated in 1996 as a series of workshops "on Satisfiability." By the third meeting in 2000, the workshop had attracted a mix of theorists and experimentalists whose common interest was the enhancement of our basic understanding of the theoretical underpinnings of the Satisfiability problem as well as the development of scalable algorithms for its solution in a wide range of application domains. In 2002 a competition of SAT solvers was inaugurated to spur further algorithmic and implementation developments, and to create an eclectic collection of benchmarks. The competition—expanded in subsequent years to include pseudo Boolean, QBF, and MAX-SAT solvers—has become an integral part of these meetings, adding an element of excitement and anticipation. The interplay between theory and application, as well as the increased interest in Satisfiability from a wider community of researchers, led to the natural evolution of these initial workshops into the current conference format. The annual SAT conference is now universally recognized as "the venue" for publishing the latest advances in SAT research.

This year marks the tenth SAT meeting. SAT is now interpreted in a broad sense to include not just propositional satisfiability, but also pseudo-Boolean constraint solving and optimization (PB), quantified Boolean formulae (QBF), constraint programming techniques (CP) for word-level problems and their propositional encoding, and satisfiability modulo theories (SMT). Submissions were solicited for original research on proof systems and proof complexity, search algorithms and heuristics, analysis of algorithms, hard instances, randomized formulae, problem encodings, industrial applications, solvers, simplifiers and tools, case studies and empirical results. A total of 74 submissions were received and rigorously reviewed by a 35-member international Technical Program Committee (TPC), with each paper receiving at least four independent reviews. Of these submissions, the TPC decided to accept 22 as regular papers (14 pages, 25-minute presentation) and 12 as short papers (6 pages, 12-minute presentation). The accepted papers were organized into nine sessions and their full text is included in these proceedings.

The conference program also featured two invited presentations. The first, by Martin Davis, chronicled the original development of the "DPLL" algorithm and proposed an unorthodox take on the P=NP problem. The second, by Andrei Voronkov, addressed new encodings that enable succinct representations of certain combinatorial problems in the Bernays – Schonfinkel fragment of first-order logic.

A number of additional events were associated with the SAT conference, including the SAT competition, the QBF evaluation, the PB evaluation, the MAX-SAT evaluation, and a special session on trends in modern SAT solvers.

We would like to acknowledge several people for their help: the SAT Local Chair, Ines Lynce; the organizers of the SAT competition, Daniel Le Berre, Laurent Simon, Ewald Speckenmeyer, Geoff Sutcliffe and Lintao Zhang; the organizers of the QBF evaluation, Massimo Narizzano, Luca Pulina and Armando Tacchella; the organizers of the PB evaluation, Vasco Manquinho and Olivier Roussel; and finally the organizers of the Max-SAT evaluation, Josep Argelich, Chu-Min Li, Felip Manyà and Jordi Planes. Last, but not least, we thank the Program Committee and the additional external reviewers for their careful and thorough work, without which it would not have been possible for us to put together such a high-quality conference program.

We also thank Andrei Voronkov for the EasyChair system. EasyChair was instrumental in handling of paper submissions, paper reviewing, paper discussion, and assembly of the proceedings. Finally, we would like to thank the following sponsors for their generous support of SAT 2007: Cadence Design Systems, Cornell's Intelligent Information Systems Institute, Intel Corporation, Luso-American Foundation, Magma Design Automation, Microsoft Corporation, NEC Laboratories, and Synopsys Inc. A number of other institutions provided critical logistical support for managing the organization of the conference: INESC-ID, Instituto Superior Técnico, the University of Michigan, and the University of Southampton.

May 2007

Joao Marques-Silva
Karem Sakallah

Organization

Conference Chairs

Joao Marques-Silva
Karem Sakallah

Local Chair

Ines Lynce

Technical Program Committee

Fahiem Bacchus
Paul Beame
Armin Biere
Adnan Darwiche
Leonardo de Moura
Niklas Een
John Franco
Ian Gent
Enrico Giunchiglia
Carla Gomes
Aarti Gupta
Ziyad Hanna
Edward Hirsch
Joonyoung Kim
Hans Kleine-Büning
James Kukula
Oliver Kullmann
Daniel Le Berre
Chu-Min Li
Ines Lynce
Panagiotis Manolios
Vasco Manquinho
Slawomir Pilarski
Steve Prestwich
Roberto Sebastiani
Hossein Sheini
Laurent Simon
Ewald Speckenmeyer
Ofer Strichman
Stefan Szeider
Armando Tacchella
Allen Van Gelder
Hans van Maaren
Toby Walsh
Lintao Zhang

External Reviewers

Dimitris Achlioptas
Johan Alfredsson
Fadi Aloul
Anbulagan Anbulagan
Josep Argelich
Gilles Audemard
Ritwik Bhattacharya
Jesse Bingham
Per Bjesse
Nikolaj Bjorner
Roberto Bruttomesso
Uwe Bubeck
Arthur Choi
Alessandro Cimatti
Stefan Dantchev
Jessica Davies
Gilles Dequen
Laure Devendeville
Peter Dillinger
Kutsy Ekaterina
Yulik Feldman
Anders Franzen
Zhaohui Fu
Roman Gershman
Eugene Goldberg
Dan Goldwasser
Alberto Griggio
Marijn Heule
Jinbo Huang
Dmitry Itsykson
Attila Jurecska
Toni Jussila
Zurab Khasidashvili
Matthew Kitching
Arist Kojevnikov
Andrei Krokhin
Alexander Kulikov
Elitza Maneva
Felip Manya

Marco Maratea
Igor Markov
Arie Matsliah
Bertrand Mazure
Thomas Meyer
Alan Mishchenko
António Morgado
Alexander Nadel
Naren Narasimhan
Massimo Narizzano
Peter Nightingale
Sergey Nikolenko
Cedric Piette
Knot Pipatsrisawat
Stefan Porschen
Olivier Roussel
Bert Randerath
Federico Ricci-Tersenghi
Michael Ryavchev
Vadim Ryvchin
Ashish Sabharwal
Marko Samer
Horst Samulowitz
Tian Sang
Carsten Sinz
Sudarshan Srinivasan
Ted Stanion
Baruch Sterin
Peter Stuckey
Niklas Sörensson
Heather Trumbower
Bubeck Uwe
Michael Veksler
Michele Vescovi
Daron Vroon
Sean Weaver
Wanxia Wei
Jesse Whittemore
Hans Zantema

Sponsoring Institutions

Cadence Design Systems
Intel Corp.
Intelligent Information Systems Institute, Cornell
Luso-American Foundation
Magma Design Automation
Microsoft Research
NEC Research Laboratories
Synopsys Inc.
INESC-ID
Instituto Superior Técnico
The University of Michigan
The University of Southampton

Table of Contents

SAT: Past and Future 1
Martin Davis

Encodings of Problems in Effectively Propositional Logic 3
Juan Antonio Navarro-Pérez and Andrei Voronkov

Efficient Circuit to CNF Conversion 4
Panagiotis Manolios and Daron Vroon

Mapping CSP into Many-Valued SAT 10
Carlos Ansótegui, María Luisa Bonet, Jordi Levy, and Felip Manyà

Circuit Based Encoding of CNF Formula 16
Gilles Audemard and Lakhdar Saïs

Breaking Symmetries in SAT Matrix Models 22
Inês Lynce and Joao Marques-Silva

Partial Max-SAT Solvers with Clause Learning 28
Josep Argelich and Felip Manyà

MiniMaxSat: A New Weighted Max-SAT Solver 41
Federico Heras, Javier Larrosa, and Albert Oliveras

Solving Multi-objective Pseudo-Boolean Problems 56
Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich

Improved Lower Bounds for Tree-Like Resolution over Linear Inequalities 70
Arist Kojevnikov

Horn Upper Bounds and Renaming 80
Marina Langlois, Robert H. Sloan, and György Turán

Matched Formulas and Backdoor Sets 94
Stefan Szeider

Short XORs for Model Counting: From Theory to Practice 100
Carla P. Gomes, Joerg Hoffmann, Ashish Sabharwal, and Bart Selman

Variable Dependency in Local Search: Prevention Is Better Than Cure 107
Steven Prestwich

Combining Adaptive Noise and Look-Ahead in Local Search for SAT ... 121
Chu Min Li, Wanxia Wei, and Harry Zhang

From Idempotent Generalized Boolean Assignments to Multi-bit Search ... 134
Marijn Heule and Hans van Maaren

Satisfiability with Exponential Families ... 148
Dominik Scheder and Philipp Zumstein

Formalizing Dangerous SAT Encodings ... 159
Alexander Hertel, Philipp Hertel, and Alasdair Urquhart

Algorithms for Variable-Weighted 2-SAT and Dual Problems ... 173
Stefan Porschen and Ewald Speckenmeyer

On the Boolean Connectivity Problem for Horn Relations ... 187
Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto

A First Step Towards a Unified Proof Checker for QBF ... 201
Toni Jussila, Armin Biere, Carsten Sinz, Daniel Kröning, and Christoph M. Wintersteiger

Dynamically Partitioning for Solving QBF ... 215
Horst Samulowitz and Fahiem Bacchus

Backdoor Sets of Quantified Boolean Formulas ... 230
Marko Samer and Stefan Szeider

Bounded Universal Expansion for Preprocessing QBF ... 244
Uwe Bubeck and Hans Kleine Büning

Effective Incorporation of Double Look-Ahead Procedures ... 258
Marijn Heule and Hans van Maaren

Applying Logic Synthesis for Speeding Up SAT ... 272
Niklas Een, Alan Mishchenko, and Niklas Sörensson

Towards a Better Understanding of the Functionality of a Conflict-Driven SAT Solver ... 287
Nachum Dershowitz, Ziyad Hanna, and Alexander Nadel

A Lightweight Component Caching Scheme for Satisfiability Solvers ... 294
Knot Pipatsrisawat and Adnan Darwiche

Minimum 2CNF Resolution Refutations in Polynomial Time ... 300
Joshua Buresh-Oppenheim and David Mitchell

Polynomial Time SAT Decision for Complementation-Invariant Clause-Sets, and Sign-non-Singular Matrices ... 314
Oliver Kullmann

Verifying Propositional Unsatisfiability: Pitfalls to Avoid 328
Allen Van Gelder

A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories 334
Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani

SAT Solving for Termination Analysis with Polynomial Interpretations 340
Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl

Fault Localization and Correction with QBF 355
Stefan Staber and Roderick Bloem

Sensor Deployment for Failure Diagnosis in Networked Aerial Robots: A Satisfiability-Based Approach 369
Fadi A. Aloul and Nagaragan Kandasamy

Inversion Attacks on Secure Hash Functions Using SAT Solvers 377
Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan

Author Index 383

SAT: Past and Future

Martin Davis

Part I. Davis-Putnam: An Accidental Algorithm

During the summer of 1957, Hilary Putnam and I, both junior faculty, were attending an unprecedented month-long "institute" devoted to logic at Cornell University along with 82 other logicians. Our families were sharing a house and the two of us were together every day working together and separately on a number of things, but not on the satisfiability problem. After we had made some progress towards a negative solution of Hilbert's 10th Problem (H10: the question of the existence of an algorithm for determining whether a given polynomial equation has an integer solution), we were eager to continue collaborating. Our idea was to seek funding through my institution which was a branch of Rensselaer Polytechnic in Eastern Connecticut so Hilary and his family could escape steamy summers in Princeton for the attractive lakeside accommodations available in my locale. Not believing that anyone would pay us to work on H10, considered a super long shot, we patched together a proposal to investigate procedures for theorem-proving in first-order logic. Because it was too late for the usual funding agencies, following a tip we submitted our proposal to the National Security Agency. They funded it on condition that our report *not* mention them, and that we forget about first-order logic, and just concentrate on satisfiability. Our report, which was submitted at the end of the summer of 1958, contained all the procedures that were eventually combined in the algorithms later designated as DP and DPLL. During the summer of 1959, we were supported by the US Air Force Office of Scientific Research. We worked very hard on H10 and made some significant progress. But because our proposal had emphasized theorem-proving procedures, we hastily concocted one using some of the work from the previous summer, and submitted it to the JACM. That was the origin of Davis-Putnam. After I moved to New York, I wanted to see our procedure implemented, and NYU put two very talented student programmers at my disposal for the purpose: Donald Loveland (who later became one of my first doctoral students) and George Logemann. The crude search we implemented led to satisfiability questions involving thousands of clauses and the original DP swamped the memory of the IBM 704. So we replaced the "rule for eliminating propositional variables" (i.e. ground binary resolution) with the splitting rule giving the algorithm a "divide and conquer" form with instances waiting to be processed swapped out onto a tape. This was the DPLL algorithm.

Part II. SAT $\notin P$?

Although everyone seems to believe that $P \neq NP$, the evidence is scant and somewhat circular. There is the fact that the problems for which good feasible worst-case algorithms are known, are solvable in poly-time. But in practice, "poly-time" really means $O(n \log n)$ or maybe $O(n^2)$ and with a manageable multiplicative constant. No-one would regard an algorithm that runs in time $10^{10^{10}} n^2$ or $O(n^{1000})$ as "feasible". But

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 1–2, 2007.

it is only the identification of poly-time computability with feasibility, in analogy with the identification of Turing computability with effective computability, that makes the existence of so many NP-complete problems seem to be evidence for $P \neq NP$. If there are horrendous poly-time algorithms for these NP-complete problems, how might we come to know it? Is anyone seeking such algorithms? Theorists have built their poly-time hierarchy in stages mimicking the arithmetic hierarchy of the logicians with P at the bottom and P-SPACE at the top. But they have been unable to prove a single separation theorem between the levels. For all we know, the entire edifice could collapse with $P = P$-SPACE. Computer science is a very young subject. Mathematicians know from hard experience that problems easy to state can take hundreds of years to resolve. But theorists blithely conclude from an implication $A \Rightarrow P = NP$ that the proposition A must be false. The case of linear programming provides a good example which can well resonate with experts on SAT. The very useful simplex method runs in exponential time in the worst case. It was thought for years that there is no poly-time algorithm for linear programming. Experts were astounded when it turned out that in fact poly-time algorithms for linear programming do exist. However, ironically enough in practice the old reliable exponential-time simplex method does better than these poly-time time algorithms. So what do we know about the question: Is SAT $\in P$? Almost nothing! It could go either way. But if it should turn out that the answer is "Yes", that would of course imply that $P = NP$, and so would entitle the person who succeeded in proving it to receive the million dollar prize the Clay Institute of Mathematics is offering. If I were 60 years younger, I'd be tempted to try!

Encodings of Problems in Effectively Propositional Logic

Juan Antonio Navarro-Pérez and Andrei Voronkov

The University of Manchester
School of Computer Science
{navarroj,voronkov}@cs.manchester.ac.uk

Solving various combinatorial problems by their translation to the propositional satisfiability problem has become commonly accepted. By optimising such translations and using efficient SAT solvers one can often solve hard problems in various domains, such as formal verification and planning.

This approach to solving combinatorial problems is usually implemented by a translation procedure turning a formal description of the problem written in a domain-specific language L (for example, SMV for model checking problems [3] or STRIPS [2] for planning problems) into a SAT problem. Such translation procedures share the following common features:

1. They contain many isomorphic or nearly isomorphic subsets of clauses obtained by the translation of the same expression of L.
2. The size of the resulting SAT problem is dominated by these copies.

In this talk the second author will present encodings able to specify some combinatorial problems, namely LTL bounded model checking [1] and planning within the Bernays-Schönfinkel fragment of first-order logic. This fragment, which also corresponds to the category of effectively propositional problems (EPR) of the CASC system competitions [4], allows a natural and succinct representation of both the transition systems corresponding to the problems and the property that one wants to verify, while avoiding the problem of creating isomorphic copies.

Our technique provides a rich collection of benchmarks with close links to real-life applications for the automated reasoning community and may boost development of new translation techniques and solvers for effectively propositional problems.

References

[1] A. Biere, A. Cimatti, E.M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207, 1999.

[2] R. Fikes and N.J. Nilsson. A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[3] K.L. McMillan. *Symbolic Model Checking*. Kluwer, 1993.

[4] G. Sutcliffe and C.B. Suttner. The state of CASC. *AI Communications*, 19(1):35–48, 2006.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, p. 3, 2007.

Efficient Circuit to CNF Conversion

Panagiotis Manolios and Daron Vroon

College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332, USA
http://www.cc.gatech.edu/home/{manolios,vroon}

Abstract. Modern SAT solvers are proficient at solving Boolean satisfiability problems in Conjunctive Normal Form (CNF). However, these problems mostly arise from general Boolean circuits that are then translated to CNF. We outline a simple and expressive data structure for describing arbitrary circuits, as well as an algorithm for converting circuits to CNF. Our experimental results over a large benchmark suite show that the CNF problems we generate are consistently smaller and more quickly solved by modern SAT solvers than the CNF problems generated by current CNF generation methods.

1 Introduction

The recent drastic improvements to SAT solving technology have led to its wide applicability in domains ranging from hardware and software verification to computational biology to AI planning. While the actively developed SAT solvers overwhelmingly require input to be in CNF, most of the applications that use SAT technology have problems that are more naturally expressed as Boolean combinational circuits. In order to use current SAT solvers, users are therefore required to generate CNF and they tend to do this using variants of the Tseitin algorithm, *e.g.*, this is the case with Barcelogic Tools [8] and Yices [2].

In this paper we introduce NICE dags, a new data structure for representing circuits. We also introduce a new algorithm for translating from NICE dags to CNF. We have compared our algorithm to both the Tseitin algorithm and to Jackson and Sheridan's state-of-the-art algorithm, using a benchmark suite containing over 8,000 problems from various domains. Our extensive evaluations show that the translation from circuits to CNF can significantly impact overall SAT solving time and that our algorithm leads to significant time savings over the other two approaches. In fact, for numerous problems that our approach can easily handle, both algorithms generate CNF on which SAT solvers time out.

The work reported in this paper is implemented in the publicly available Bit-level Analysis Tool (BAT) [7]. BAT provides a high-level, feature-rich, type-safe language that includes user-defined functions, arbitrary sized bit vectors and operations on them, existential arrays, etc. With the results reported in this paper, we are able to provide potential users of SAT technology a much higher-level interface than current CNF-based SAT solvers. This makes it much easier to experiment with, use, and deploy SAT technology, while still being able to take advantage of improvements to the actively developed SAT solvers, which are mostly CNF-based.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 4–9, 2007.

Fig. 1. Two examples of NICE dags

2 Related Work

Most modern CNF conversion algorithms are variations of the one originally discovered by Tseitin [9]. This algorithm converts dags composed of internal vertices labeled with $\wedge$, $\vee$, and $\neg$, as well as external vertices labeled with variable names. The algorithm introduces a new variable, x, for each internal node, v that is a child of a $\vee$ node, and adds the constraint $x \leftrightarrow v$ to the original SAT problem. This is a linear algorithm.

The Jackson-Sheridan algorithm for conversion to CNF was introduced in 2004 [5]. It takes a Reduced Boolean Circuit (RBC) as input, and is built around a heuristic that is used to decide when to introduce variables for internal vertices. The aim is to minimize the number of clauses in the resulting CNF. However, $\leftrightarrow$ vertices are rewritten into conjunctions of disjunctions before conversion into CNF, which results in the loss of information that can lead to the generation of fewer intermediate variables. The algorithm is quadratic.

Brummayer and Biere have recently introduced an algorithm for translating from AIGs (And-Inverter Graphs) to CNF [1]. The techniques used are mostly orthogonal to ours, and, for future work, it would be interesting to explore the incorporation of some of their ideas into our algorithm.

Another CNF translation that deals directly with *ite* vertices is presented by Velev in [10]. Our algorithm subsumes this algorithm, *e.g.*, we merge nested *ite* vertices in all the ways that Velev's algorithm does and more. Also, unlike Velev's algorithm, we constrain intermediate variables with implications rather than equivalences, which significantly reduces the number of clauses created when a new variable is introduced.

3 NICE Dags

The circuit representation that our CNF translation algorithm takes as input is the *Negation, Ite, Conjuction, and Equivalence dag (NICE dag)*, which contains external variable nodes, and as the name suggests, internal nodes labeled with $\neg$, $\leftrightarrow$, $\wedge$, and *ite*. The three outgoing edges of an *ite* are labeled with *test*, *then*, or *else*, as appropriate. As with RBCs, NICE dags are further constrained to maximize sharing. For example, no two nodes in the dag are allowed to have the same label and children, and no *ite* node can have a *test* or *then* child labeled with $\neg$. Two example NICE dags are given in Figure 1.

```
cnf ((V, E))
  pseudo-expand ((V, E))
  count-shares ((V, E))
  CL := ∅
  for all v ∈ V do
    clauses+ (v) := null
    clauses- (v) := null
  C := cnf+ (source((V, E)))
  return C ∪ CL
```

Fig. 2. Main CNF conversion function

4 CNF Conversion

In order to understand our CNF conversion algorithm, it is important to understand the following terminology. The *number of shares* of a node is the number of incoming edges the node has. A *path* is a sequence of vertices, $v_1, v_2, \ldots, v_n$ such that v_1 is the root of the dag and each consecutive pair of vertices forms an edge. The path polarity of a given path is *negative* if it contains an odd number of $\neg$ nodes, and *positive* otherwise. The *number of negative (positive) shares* for a node is the number of predecessors of the node that are the last vertex in a path of negative (positive) polarity. Note that a predecessor vertex can appear in a path of negative polarity and a path of positive polarity, so the the number of negative and positive shares can add up to more than the total number of shares.

Our central CNF conversion algorithm is given in Figure 2. It takes a NICE dag as input and returns a set of clauses that are equisatisfiable to the input. The algorithm begins with the *pseudo-expansion* of the the *ite* and $\leftrightarrow$ nodes of the NICE dag. We will return to this function momentarily.

The next function, `count-shares`, marks each node with its number of negative and positive shares. This is followed by initialization. A global variable, *CL*, is initialized to $\emptyset$. *CL* will contain the clauses constraining the variables introduced for internal vertices. All of the vertices, $v \in V$ are marked with a `clauses+` and `clauses-` value of *null*. These will eventually contain the clauses generated for v and $\neg v$, respectively.

The core of the algorithm, which is too long to give here, consists of two functions, `cnf+` and `cnf-`, which take a vertex, v, and compute `clauses+(`v`)` and `clauses-(`v`)` respectively. These functions recursively visit the children of v to compute their clause representations, and then combine the resulting clause sets in the appropriate way. For example, when applied to a $\wedge$ node, u, `cnf+` calls `cnf+` on all the children of u and then unions their clause lists together to form the clause list for u. A $\neg$ node, w, is processed by `cnf+` (`cnf-`) by applying `cnf-` (`cnf+`) to its child. Another way to see this is that `cnf+` and `cnf-` process the NICE dag using depth-first search, keeping track of the polarity of the path that led to the current node, and post-processing the node accordingly. The `cnf+` and `cnf-` algorithms have the following key features:

- The decision to introduce variables is made separately for a vertex and its negation, and the variable is constrained with an implication rather than an equivalence if it only represents a vertex or its negation, and not both. For example, if a variable, x is introduced for v, then the constraint $x \rightarrow v$ is added to CL. If we later decide that a variable is needed for $\neg v$, we add the constraint $\neg x \rightarrow \neg v$.
- When forming disjunctions, we use a similar heuristic to Jackson-Sheridan for deciding when to introduce new variables [5].
- A variable is always introduced for v ($\neg v$) if the number of positive (negative) shares for v is more than 1 and $|\texttt{clauses+}(v)| > 1$ ($|\texttt{clauses-}(v)| > 1$).
- *ite* and $\leftrightarrow$ are interpreted as conjunctions of disjunctions regardless of their polarity. For example, $\texttt{cnf+(if}\ u\ \texttt{then}\ v\ \texttt{else}\ w\texttt{)}$ is computed by computing $\texttt{cnf+}((\neg u \vee v) \wedge (u \vee w))$ and $\texttt{cnf-(if}\ u\ \texttt{then}\ v\ \texttt{else}\ w\texttt{)}$ is computed by computing $\texttt{cnf+}((\neg u \vee \neg v) \wedge (u \vee \neg w))$. This leads to a smaller CNF translation.

Note that this last case introduces some extra complexity to the algorithm. On the one hand, we want to translate *ite* and $\leftrightarrow$ nodes, as well as their negations, as conjunctions of disjunctions. This means that the translations of these nodes and their negations are no longer syntactic negations of each other. That is, the negation of an *ite* node, does not simply translate to a $\neg$ node whose child is the positive translation of the *ite*, since this would be a disjunction of conjunctions. That is why we do not expand *ite* and $\leftrightarrow$ nodes before the translation. This way, we can keep track of the fact that these translations are negations of each other by maintaining `clauses+` and `clauses-` for the original *ite* or $\leftrightarrow$ node.

On the other hand, in the example above, what if $u \vee w$ appears elsewhere in the dag? In this case, we will lose some sharing information if we do not expand the *ite* node before CNF conversion, since we will have another instance of $u \vee w$ when we do expand it. This is why we have the `pseudo-expand` function that starts our CNF translation algorithm. This function will mark the example above with new "pseudo-arguments" called `args+` and `args-`. The `args+` mark is set to the pair of vertices representing $(\neg u \vee v), (u \vee w)$. The `args-` mark is set to the pair of vertices representing $(\neg u \vee \neg v), (u \vee \neg w)$. Then, when counting shares, we count the shares of the "pseudo-arguments" of *ite* and $\leftrightarrow$ formulas rather than the actual arguments. This allows us to capture the appropriate sharing information without losing the negation information for *ite* and $\leftrightarrow$ and nodes.

In general, $\texttt{cnf}\ ((V, E))$ has a running time of $O(|V|^2)$. However, as we will see in our experimental results, our algorithm runs faster than Tseitin's algorithm, which is linear, showing that our algorithm is linear is practice, at least for the benchmarks tested.

5 Experimental Evaluation

We implemented Tseitin's algorithm [9], Jackson and Sheridan's algorithm [5], and our algorithm in the Bit-level Analysis Tool (BAT). BAT is a tool for solving

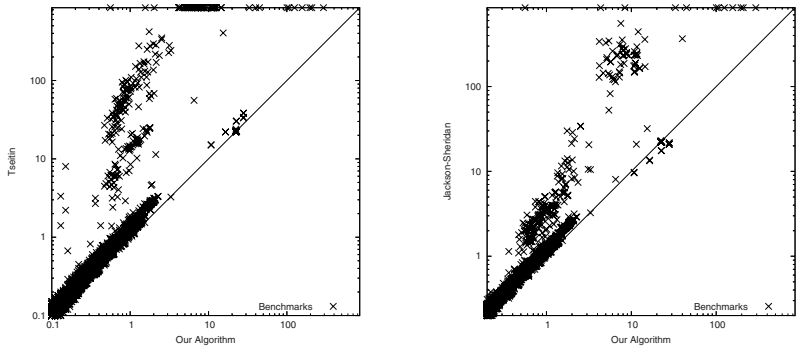


Fig. 3. Scatter plots comparing our algorithm to Tseitin and Jackson-Sheridan

formulas over the theory of bit vectors and existential arrays [7,6]. After processing arrays, BAT converts the resulting circuit into a NICE dag, which it uses to perform further simplifications. Finally, it converts the NICE dag to CNF.

We ran all three algorithms over a suite composed of 8,284 benchmarks. These include refinement theorems for two, three, and five stage pipeline machines, a correctness theorem for the Instruction Cache RAM unit from the Sun PicoJava II microprocessor, a theorem involving the out-of-order retirement of instructions, and benchmarks from the 2006 SMT competition for the quantifier-free theory of uninterpreted functions over 32-bit bit-vectors. We used the latest version of MiniSat2 with simplification enabled to solve the resulting CNF problems [4]. The BAT time never exceeded 25 seconds for any benchmark, and MiniSat2 was given a timeout of 10 minutes. Experiments were run on an 2.4 GHz Intel Pentium 4 machine with a 512K cache and 1 GB of memory.

Scatter plots comparing our algorithm to the Tseitin and Jackson-Sheridan algorithms are given in Figure 3. Times reported are the total time taken for BAT and MiniSat2. Represented here are all benchmarks that took at least 0.1 second for Tseitin or Jackson-Sheridan to complete. Those taking less than a 0.1 second had comparable running times for all algorithms. Points above the diagonal indicate that our algorithm resulted in a faster overall solving time. Points along the top edge indicate time-outs for the competing algorithm. Based on these numbers, our algorithm clearly out-performs Tseitin and Jackson-Sheridan on almost all problems, often by orders of magnitude. Since this improvement is observed in the presence of CNF preprocessing (as performed by Minisat2), our view is that preprocessing techniques are not a satisfactory alternative to CNF translation [3]; rather these two approaches can be complementary.

Also, as noted earlier, despite the fact that our CNF translation is quadratic in the worst case, in practice it is faster that the linear-time Tseitin translation. The total time spent translating the benchmarks to CNF was 3,725 seconds for Tseitin and 3,409 seconds for our algorithm.

6 Conclusions

We presented the notion of NICE dags, a data structure that can be used to represent arbitrary circuits and outlined an algorithm that converts NICE dags to CNF. As our extensive experiments on over 8,000 benchmark problems show, our algorithm leads to very large efficiency gains in total SAT times over both an efficient variant of the widely used Tseitin translation algorithm and the Jackson-Sheridan algorithm, even in the presence of CNF preprocessing.

References

1. R. Brummayer and A. Biere. Local two-level and-inverter graph minimization without blowup. In *Proc. 2nd Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS '06)*, October 2006.
2. B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In T. Ball and R. B. Jones, editors, *Computer Aided Verification, CAV 2006*, volume 4144 of *LNCS*, pages 81–94, 2006.
3. N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In F. Bacchus and T. Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005*, volume 3569 of *LNCS*, pages 61–75. Springer, 2005.
4. N. Eén and N. Sörensson. MiniSat - a SAT solver with conflict-clause minimization. In F. Bacchus and T. Walsh, editors, *Posters of the 8th international Conference on Theory and Applications of Satisfiability Testing*, 2005.
5. P. Jackson and D. Sheridan. Clause form conversions for Boolean circuits. In H. H. Hoos and D. G. Mitchell, editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004*, volume 3542 of *LNCS*, pages 183–198. Springer, 2004.
6. P. Manolios, S. K. Srinivasan, and D. Vroon. Automatic memory reductions for RTL-level verification. In *ICCAD 2006, ACM-IEEE International Conference on Computer Aided Design*. ACM, 2006.
7. P. Manolios, S. K. Srinivasan, and D. Vroon. BAT: The Bit-level Analysis Tool. 2006. Available from `http://www.cc.gatech.edu/~manolios/bat/`.
8. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In A. Biere and C. P. Gomes, editors, *9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06*, volume 4121 of *LNCS*, pages 156–169. Springer, 2006.
9. G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part2*, pages 115–125. Consultants Bureau, New York-London, 1962.
10. M. N. Velev. Efficient translation of boolean formulas to cnf in formal verification of microprocessors. In *ASP-DAC '04: Proceedings of the 2004 conference on Asia South Pacific design automation*, pages 310–315, 2004. IEEE Press.

Mapping CSP into Many-Valued SAT*

Carlos Ansótegui[1], María Luisa Bonet[2], Jordi Levy[3], and Felip Manyà[1]

[1] Universitat de Lleida (DIEI, UdL)
[2] Universitat Politècnica de Catalunya (LSI, UPC)
[3] Artificial Intelligence Research Institute (IIIA, CSIC)

Abstract. We first define a mapping from CSP to many-valued SAT which allows to solve CSP instances with many-valued SAT solvers. Second, we define a new many-valued resolution rule and prove that it is refutation complete for many-valued CNF formulas and, moreover, enforces CSP (i, j)-consistency when applied to a many-valued SAT encoding of a CSP. Instances of our rule enforce well-known local consistency properties such as arc consistency and path consistency.

1 Introduction

SAT and CSP are problem solving paradigms which have been shown to be competitive in a wide range of domains. Both the SAT community and the CSP community have devised a number of solving techniques that have been incorporated into state-of-the-art solvers. SAT techniques are better than CSP techniques for some problems, and vice versa. In this paper, we focus on inference and our goal is to explore how CSP inference can be defined in a way similar to SAT inference, which is usually defined via resolution-like inference rules. To this end, we use the formalism provided by the many-valued clausal forms known as signed CNF formulas, and define a number of resolution rules that enforce the most important local consistency properties defined in the literature.

First, we define a mapping from CSP to signed-SAT, which is the satisfiability problem of the many-valued clausal forms known as signed CNF formula [BHM00]. A CSP instance is now represented as a list of clauses, where each clause represents a no-good of a constraint. We use signed-SAT instead of SAT to capture in a natural way the information provided by the domains of CSP variables. Second, we define a new resolution rule and prove that it is refutation complete for signed CNF formulas and, moreover, enforces (i, j)-consistency when applied to a signed-SAT encoding of a CSP. Third, we show how instances of the (i, j)-consistency rule enforce well-known local consistency properties such as arc consistency and path consistency.

The fact of reformulating the main CSP consistency properties as resolution-like inference rule has some advantages: (i) they are easier to understand, at least for the SAT community; (ii) the machinery and techniques for resolution

* Research partially supported by projects iDEAS (TIN2004-04343), Mulog (TIN2004-07933-C03-01/03) and IEA (TIN2006-15662-C02-02) funded by the MEC.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 10–15, 2007.

developed by the automated deduction community can be easily applied to CSP; (iii) CSP and SAT inference can be compared by restricting to domains of cardinality two; and (iv) a signed-SAT solver allowing to apply different resolution rules at each node of the search tree provides a framework for analysing CSP local consistency, as well as for comparing SAT and CSP inference and eventually devise new solvers.

Our work is closely related to previous attempts to understand the relation between CSP and SAT, and vice versa (see [BHW04, Gen02, Wal00]). The advantage of our approach is the use of a formalism in which we can reformulate the inference of both SAT and CSP, instead of mapping CSP into SAT and SAT into CSP as in [AM04, BHM99, BHW04, FP01].

The results of this paper can provide new insights to the existing results about exploiting the structure of CSPs into SAT solvers [ALM03, Bac06, DS06].

2 Preliminaries

2.1 Signed CNF Formulas

Definition 1. *A* truth value set*, or* domain*, N is a non-empty finite set. A* sign *is a subset $S \subseteq N$ of truth values. The* complement *of a sign S, denoted by $\overline{S}$, is $N \setminus S$. A* signed literal *is an expression of the form $S{:}x$, where S is a sign and x is a propositional variable. The set S is also called the* support *of x. The* complement *of a signed literal l of the form $S{:}x$, denoted by $\overline{l}$, is $\overline{S}{:}x$. A* signed clause *is a disjunction of signed literals. A* signed CNF formula *is a set of signed clauses (or a conjunction of clauses).*

Definition 2. *An* assignment *for a signed CNF formula is a mapping that assigns to every propositional variable an element of the truth value set.*
An assignment I satisfies *a signed literal $S{:}x$, if $I(x) \in S$. It* satisfies *a signed clause C, if it satisfies at least one of the signed literals in C. It* satisfies *a signed CNF formula Γ, if it satisfies all clauses in Γ.*

A signed CNF formula is satisfiable*, if it is satisfied by at least one assignment; otherwise it is* unsatisfiable*. The* signed-SAT problem *for a signed CNF formula ϕ consists of determining whether ϕ is satisfiable.*

We give now two refutationally complete inference systems for signed-SAT. The first one is defined by the next two rules on the left [Häh93], while the second one is defined by the rule on the right [Häh94].

Signed Binary Resolution

$$\frac{S{:}x \vee A \qquad S'{:}x \vee B}{S \cap S'{:}x \vee A \vee B}$$

Simplification

$$\frac{\emptyset{:}x \vee D}{D}$$

Signed Parallel Resolution

$$\frac{S_1{:}x \vee A_1 \quad \cdots \quad S_k{:}x \vee A_k}{A_1 \vee \cdots \vee A_k} \quad \text{whenever } \bigcap_{i=1}^{k} S_i = \emptyset$$

Also we assume w.l.o.g. that every variable in a clause appears only once collapsing different occurrences of a literal making the union of the supports.

2.2 Constraint Satisfaction Problems

Definition 3. *A* constraint satisfaction problem (CSP) *instance, or* constraint network*, is defined as a triple* $\langle X, D, C\rangle$*, where* $X = \{x_1, \ldots, x_n\}$ *is a set of variables,* $D = \{d(x_1), \ldots, d(x_n)\}$ *is a set of domains containing the values the variables may take, and* $C = \{C_1, \ldots, C_p\}$ *is a set of constraints. Each constraint* $C_i = \langle S_i, R_i\rangle$ *is defined as a relation* R_i *over a subset of variables* $S_i = \{x_{i_1}, \ldots, x_{i_k}\}$*, called the* constraint scope*. The relation* R_i *may be represented extensionally as a subset of the Cartesian product* $d(x_{i_1}) \times \cdots \times d(x_{i_k})$*.*

Definition 4. *An* assignment *for a CSP instance* $\langle X, D, C\rangle$ *is a mapping that assigns to each variable* $x_i \in Y$*, where* $Y \subseteq X$*, a value from* $d(x_i)$*. An assignment* I *satisfies a constraint* $\langle\{x_{i_1}, \ldots, x_{i_k}\}, R_i\rangle \in C$*, if* $\langle I(x_{i_1}), \ldots, I(x_{i_k})\rangle \in R_i$*. An assignment* I *over the set of variables* Y *is consistent, if for every constraint* $C_i \in C$ *defined on variables* $Y' \subseteq Y$*,* I *restricted to* Y' *satisfies* C_i*.*

The Constraint Satisfaction Problem (CSP) consists of, given a CSP instance, finding an assignment that satisfies the instance, if it exists, or showing that it is unsatisfiable.

We next define the main local consistency properties that have been defined in the literature.

Definition 5. *A CSP is* (i, j)-consistent*, for* $i \geq 0$ *and* $j \geq 1$*, if it has non-empty domains and any consistent instantiation of* i *variables can be extended to a consistent instantiation of* j *additional variables.*

A CSP is node consistent *if it is* $(0, 1)$*-consistent, it is* arc consistent *if it is* $(1, 1)$*-consistent, and it is* path consistent *if it is* $(2, 1)$*-consistent.*

A CSP is k-consistent*, for* $k \geq 1$*, if it is* $(k - 1, 1)$*-consistent.*

A CSP is strong k-consistent*, for* $k \geq 1$*, if it is* i*-consistent for every* $i \in \{1, \ldots, k\}$*.*

2.3 Mapping CSP into Signed-SAT

We define a mapping that translates a CSP instance P into a signed-SAT instance P' in such a way that P is satisfiable iff P' is satisfiable [ABLM07]. The encoding basically translates no-goods into clauses.

Definition 6. *The* signed encoding *of a CSP instance* $\langle X, D, C\rangle$ *is the signed CNF formula over the truth value set* $N = \bigcup_{x_i \in D} d(x_i)$ *that contains, for every constraint* $\langle\{x_1, \ldots, x_k\}, R\rangle \in C$ *and every possible tuple* $\langle b_1, \ldots, b_k\rangle \in d(x_1) \times \cdots \times d(x_k)$ *such that* $(b_1, \ldots, b_k) \notin R$*, the clause:*

$$\overline{\{b_1\}}{:}x_1 \vee \cdots \vee \overline{\{b_k\}}{:}x_k$$

Moreover, for every variable x *and every value* $b \in N$ *such that* $x \notin d(x)$*, we add the unary clause* $\overline{\{b\}}{:}x$ *.*

3 CSP Inference as Signed Resolution

In this section we define a sound and complete signed resolution rule, called signed (i,j)-consistency, that enforces CSP (i,j)-consistency when applied to a signed-SAT encoded CSP. Then, we show that instances of the rule enforce arc consistency and path consistency. The next lemma will help us understand the rule.

Lemma 1. *Let* $\phi = \{ S_{1,1}{:}y_1 \vee \cdots \vee S_{1,p}{:}y_p, \ldots, S_{k,1}{:}y_1 \vee \cdots \vee S_{k,p}{:}y_p \}$ *be a set of signed clauses. Then, the set of assignments that satisfies all the clauses of* ϕ *can be characterized by the set* $\bigcap_{r=1}^{k} \overline{\overline{S_{r,1}} \times \cdots \times \overline{S_{r,p}}}$.

Proof: The set $\overline{S_{r,1}} \times \cdots \times \overline{S_{r,p}}$ is exactly the set of assignments that falsify the clause $S_{r,1}{:}y_1 \vee \cdots \vee S_{r,p}{:}y_p$. Therefore $\overline{\overline{S_{r,1}} \times \ldots \times \overline{S_{r,p}}}$ is the set of assignments that satisfy it. As a conclusion, the set of assignments that satisfy all the clauses is $\bigcap_{r=1}^{k} \overline{\overline{S_{r,1}} \times \cdots \times \overline{S_{r,p}}}$. ■

Signed (i,j)-Consistency Rule:

$$\frac{\begin{array}{c} S_{1,1}{:}x_1 \vee \cdots \vee S_{1,i}{:}x_i \vee S_{1,i+1}{:}x_{i+1} \vee \cdots \vee S_{1,i+j}{:}x_{i+j} \\ \cdots \\ S_{k,1}{:}x_1 \vee \cdots \vee S_{k,i}{:}x_i \vee S_{k,i+1}{:}x_{i+1} \vee \cdots \vee S_{k,i+j}{:}x_{i+j} \end{array}}{\bigcup_{r=1}^{k} S_{r,1}{:}x_1 \vee \cdots \vee \bigcup_{r=1}^{k} S_{r,i}{:}x_i}$$

whenever $\bigcap_{r=1}^{k} \overline{\overline{S_{r,i+1}} \times \cdots \times \overline{S_{r,i+j}}} = \emptyset$, $i \geq 0$ and $j \geq 1$

Remark 1. Since we start with a no-good representation of the constraints, the initial clauses will have all the supports of the form $\overline{\{b\}}$, for some $b \in N$. Then, when we apply the rule, for every $l = 1, \ldots, i$, there exists a $b \in N$ such that, for all $r = 1, \ldots, k$, we have either $S_{r,l} = \overline{\{b\}}$ or $S_{r,l} = \emptyset$; otherwise the rule concludes a tautology. Therefore, in the conclusion of the rule $\bigcup_{r=1}^{k} S_{r,l}{:}x_l$ is either empty or has the form $\overline{\{b\}}$ for some $b \in N$; thus, the conclusion of the rule also preserves the no-good representation form.

In the (i,j)-consistency rule, the last j variables $x_{i+1}, \ldots, x_{i+j}$ are called *resolving variables*. In the (i,j)-consistency rule we can add the restriction that all variables appear in at least one clause ($\bigcup_{r=1}^{k} S_{r,l} \neq \emptyset$, for $l = 1, \ldots, i+j$). We call this version of the rule *non-strong*.

Lemma 2. *The signed* (i,j)*-consistency rule enforces CSP* (i,j)*-consistency, i.e. if the signed encoding of a CSP instance is closed by the* (i,j)*-consistency rule, then the CSP is* (i,j)*-consistent.*

The [non] strong signed $(i-1,1)$*-consistency rule enforces CSP [non] strong* i*-consistency.*

PROOF: Suppose that a set of clauses is closed by the rule, but its corresponding constraint network is not (i,j)-consistent. We have some tuple of i variables $\overline{x}$ and i consistent values $\overline{a}$ of their domains, and there exists also a tuple of j variables $\overline{y}$, such that $\overline{a}$ can not be extended to these new variables consistently. I.e. for any tuple of j values $\overline{b}$, the tuple of $i+j$ values $\overline{a,b}$ for the variables $\overline{x,y}$ falsifies some constraint about a subset of such variables (where at leas one of the y variables is present). Therefore, for any tuple $\langle b_1,\ldots,b_j\rangle$, the tuple $\langle a_1,\ldots,a_i,b_1,\ldots,b_j\rangle$ for $\langle x_1,\ldots,x_i,y_1,\ldots,y_j\rangle$ is not good, and there is a clause whose literals are a subset of $\overline{\{a_1\}}{:}x_1 \vee \cdots \vee \overline{\{a_i\}}{:}x_i \vee \overline{\{b_1\}}{:}y_1 \vee \cdots \vee \overline{\{b_j\}}{:}y_j$. Since the set of clauses is closed by the rule, and we have $\bigcap_{b_1\in N,\ldots,b_j\in N} \overline{\overline{\{b_1\}} \times \cdots \times \overline{\{b_j\}}} = \bigcap_{b_1\in N,\ldots,b_j\in N} \overline{\{b_1\} \times \cdots \times \{b_j\}} = \emptyset$ our set of clauses also contains a subclause of $\overline{\{x_1\}}{:}a_1 \vee \cdots \vee \overline{\{x_1\}}{:}a_i$ witch means that the tuple $\langle a_1,\ldots,a_i\rangle$ is not good for $\langle x_1,\ldots,x_i\rangle$ and this contradicts the assumption. The proof of the second part of the lemma has the same ingredients as the first. ■

Theorem 1. *The signed (i,j)-consistency rule defines a sound and complete resolution system for signed CNF formulas.*

PROOF: When $j=1$, the signed $(i,1)$-consistency rule is the signed parallel resolution rule. So, already the signed $(i,1)$-consistency rule is complete.

To see that it is a sound rule, notice that, by Lemma 1, since $\bigcap_{r=1}^{k} \overline{\overline{S_{r,i+1}} \times \cdots \times \overline{S_{r,i+j}}} = \emptyset$, the set of clauses $\{S_{1,i+1} : x_{i+1} \vee \cdots \vee S_{1,i+j} : x_{i+j}, \ldots, S_{k,i+1} : x_{i+1} \vee \cdots \vee S_{k,i+j} : x_{i+j}\}$ is unsatisfiable. By the completeness of the parallel resolution rule we can obtain the empty clause from them. Now, from this refutation we do the following transformation. We change the set of premises by $\{\, S_{1,1}{:}x_1 \vee \cdots \vee S_{1,i}{:}x_i \vee S_{1,i+1}{:}x_{i+1} \vee \cdots \vee S_{1,i+j}{:}x_{i+j}\,, \ldots,\, S_{k,1}{:}x_1 \vee \cdots \vee S_{k,i}{:}x_i \vee S_{k,i+1}{:}x_{i+1} \vee \cdots \vee S_{k,i+j}{:}x_{i+j}\,\}$. The rest of the proof is identical, but keeping the appended parts along. At this point we will not produce the empty clause, but the clause $\bigcup_{r=1}^{k} S_{r,1}{:}x_1 \vee \cdots \vee \bigcup_{r=1}^{k} S_{r,i}{:}x_i$. ■

Arc Consistency Rule:

$$\frac{\begin{array}{l}\overline{\{a\}}{:}x \vee \overline{\{j_1\}}{:}y \\ \cdots \\ \overline{\{a\}}{:}x \vee \overline{\{j_s\}}{:}y \\ \qquad\qquad \overline{\{j_{s+1}\}}{:}y \\ \qquad\qquad \cdots \\ \qquad\qquad \overline{\{j_m\}}{:}y\end{array}}{\overline{\{a\}}{:}x}$$

where $s \geq 1$ and $\{j_1,\ldots,j_m\} = N$

Path Consistency Rule:

$$\frac{\begin{array}{l}\overline{\{a\}}{:}x \vee \qquad\qquad \overline{\{j_1\}}{:}z \\ \qquad\qquad\qquad \cdots \\ \overline{\{a\}}{:}x \vee \qquad\qquad \overline{\{j_s\}}{:}z \\ \qquad\qquad \overline{\{b\}}{:}y \vee \overline{\{j_{s+1}\}}{:}z \\ \qquad\qquad\qquad \cdots \\ \qquad\qquad \overline{\{b\}}{:}y \vee \overline{\{j_r\}}{:}z \\ \qquad\qquad\qquad\quad \overline{\{j_{r+1}\}}{:}z \\ \qquad\qquad\qquad\quad \cdots \\ \qquad\qquad\qquad\quad \overline{\{j_m\}}{:}z\end{array}}{\overline{\{a\}}{:}x \vee \overline{\{b\}}{:}y}$$

where $r > s \geq 1$ and $\{j_1,\ldots,j_m\} = N$

Fig. 1. Instances of the non-strong signed (i,j)-consistency rule

In Figure 1 we present instances of the non-strong signed (i, j)-consistency rule that enforce local consistency properties like arc consistency and path consistency. We use supports of the form $\{\overline{b}\}$ given that we have already shown that this form of signs is preserved by inferences (see Remark 1). Basically, the arc consistency rule reduces the domain of the variable x to exclude the value a when it does not have support in y. The algorithm that enforces path consistency works by removing all the satisfying pairs of a constraint that cannot be extended to another variable in the way just defined.

References

[ABLM07] C. Ansótegui, M. Bonet, J. Levy, and F. Manyà. The logic behind weighted CSP. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence, IJCAI'07*, pages 32–37, 2007.

[ALM03] C. Ansótegui, J. Larrubia, and F. Manyà. Boosting Chaff's performance by incorporating CSP heuristics. In *Proc. of the 9th Int. Conf. on Principles and Practice of Constraint Programming, CP'03*, pages 96–107. Springer LNCS 2833, 2003.

[AM04] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables into problems with boolean variables. In *Proc. of the 7th Int. Conf. on Theory and Applications of Satisfiability Testing, SAT'04*, pages 1–15. Springer LNCS 3542, 2004.

[Bac06] F. Bacchus. CSPs: Adding structure to SAT. In *Proc. of the 9th Int. Conf. on Theory and Applications of Satisfiability Testing, SAT'06*, page 10. Springer LNCS 4121, 2006.

[BHM99] B. Beckert, R. Hähnle, and F. Manyà. Transformations between signed and classical clause logic. In *Proc. of the 29th Int. Symp. on Multiple-Valued Logics, ISMVL'99*, pages 248–255, 1999.

[BHM00] B. Beckert, R. Hähnle, and F. Manyà. The SAT problem of signed CNF formulas. In *Labelled Deduction*, volume 17 of *Applied Logic Series*, pages 61–82. Kluwer, Dordrecht, 2000.

[BHW04] C. Bessière, E. Hebrard, and T. Walsh. Local consistencies in SAT. In *Proc. of the 6th Int. Conf. on Theory and Applications of Satisfiability Testing, SAT'03*, pages 299–314. Springer LNCS 2919, 2004.

[DS06] Y. Dimopoulos and K. Stergiou. Propagation in CSP and SAT. In *Proc. of the 12th Int. Conf. on Principles and Practice of Constraint Programming, CP'06*, pages 137–151. Springer LNCS 4204, 2006.

[FP01] A. M. Frisch and T. J. Peugniez. Solving non-boolean satisfiability problems with stochastic local search. In *Proc. of the Int. Joint Conf. on Artificial Intelligence, IJCAI'01*, pages 282–288, 2001.

[Gen02] I. P. Gent. Arc consistency in SAT. In *Proc. of the 15th European Conf. on Artificial Intelligence, ECAI'02*, pages 121–125, 2002.

[Häh93] R. Hähnle. Short CNF in finitely-valued logics. In *Proc., Int. Symp. on Methodologies for Intelligent Systems, ISMIS'93*, pages 49–58. Springer LNCS 689, 1993.

[Häh94] R. Hähnle. Efficient deduction in many-valued logics. In *Proc. of the Int. Symp. on Multiple-Valued Logics, ISMVL'94*, pages 240–249. IEEE Press, 1994.

[Wal00] T. Walsh. SAT v CSP. In *Proc. of the 6th Int. Conf. on Principles of Constraint Programming, CP'00*, pages 441–456. Springer LNCS 1894, 2000.

Circuit Based Encoding of CNF Formula

Gilles Audemard and Lakhdar Saïs

CRIL CNRS – Université d'Artois
{audemard,sais}@cril.univ-artois.fr

Abstract. In this paper a new circuit SAT based encoding of boolean formula is proposed. It makes an original use of the concept of restrictive models introduced by Boufkhad to polynomially translate any formula in conjunctive normal form (CNF) to a circuit SAT representation (a conjunction of gates and clauses). Our proposed encoding preserves the satisfiability of the original formula. The set of models of the obtained circuit w.r.t. the original set of variables is a subset of the models (with special characteristics) of the original formula. We also provided a connection between our encoding and the satisfiability of the original formula i.e. when the input formula is satisfiable, our proposed translation delivers a full circuit formula. A new incremental preprocessing process is designed leading to interesting experimental improvements of the Minisat satisfiability solver.

1 Introduction

Propositional satisfiability (SAT) is the problem of deciding whether a boolean formula in conjunctive normal form (CNF) is satisfiable. Traditionally, most solvers work on a formula encoded in conjunctive normal form (CNF). However, encoding knowledge under CNF can flatten some structural knowledge that would be more apparent in more expressive propositional logic representation formalisms [11]. To take benefit from such structural knowledge, recent works have addressed this issue following two different paths of research. The first one use extended boolean formula for problem encoding (e.g. [11]). Whereas the second one tries to recover and/or to deduce structural knowledge from CNF encoding (e.g. [6]).

We follow the second approach which consists in detecting hidden structures of CNF formula. More precisely, based on two previous related works proposed by Purdom [8] (complementary search) to avoid search redundancies and by Boufkhad [2] on exploiting the restrictive solution (solution that has special characteristics), we propose a new and original encoding of any formula in conjunctive normal form as a conjunction of boolean functions (gates) and clauses. Each gate represents both a subset of clauses from the original CNF formula and a set of new additional clauses. Using auxiliary variables, we obtain a polynomial circuit SAT based encoding which preserves the satisfiability of the original formula. Using two restrictive variants of models, we derive a circuit SAT formula where the remaining clauses belong to a tractable class (horn or reverse-horn).

Our proposed translation delivers a circuit sat formula where the circuit part contains a set of covered gates (i.e. the clauses representing such gates appears in the original formula) and a set of derived gates from the interaction between different parts of the

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 16–21, 2007.

formula. The circuit SAT formula obtained by our encoding can be exploited in different ways. First as proposed recently, particularly when dealing with instances encoding EDA applications, one can exploit promising circuit SAT solver as in [7,11]. Secondly, SAT solvers can be used on the new CNF formula obtained from the circuit SAT formula. We can also exploit the derived circuit SAT formula to compute a strong backdoor set of variables [6,12].

2 Technical Background and Related Works

We use classical notations and definitions of the satisfiability problem. Let Σ be a CNF, $\mathcal{V}(\Sigma)$ (resp. $\mathcal{L}(\Sigma)$) denotes the set of variables (resp. literals) occurring in Σ. The set $\mathcal{L}(\Sigma)$ is the union of positive literals $\mathcal{L}^+(\Sigma)$ and negative literals $\mathcal{L}^-(\Sigma)$. For a literal $l \in \mathcal{L}(\Sigma)$, we can rewrite Σ as $(l \vee \alpha(l)) \wedge (\neg l \vee \alpha(\neg l)) \wedge \Gamma$, where $\alpha(l) = \forall_{c \in \Sigma | l \in c} \cup c - \{l\}$ (resp. $\alpha(\neg l) = \forall_{c \in \Sigma | \neg l \in c} \cup c - \{\neg l\}$) and $\Gamma = \{c | c \in \Sigma, c \cap \{l, \neg l\} = \emptyset\}$. We define $\Sigma \wedge x$ noted $\Sigma(x)$ as a formula obtained from Σ by assigning x the truth-value *true*. Formally $\Sigma(x) = \{C | C \in \Sigma, \{x, \neg x\} \cap C = \emptyset\} \cup \{C \backslash \{\neg x\} | C \in \Sigma, \neg x \in C\}$. A *(boolean) gate* is an expression of the form $y = f(x_1, \ldots, x_k)$, where f is a standard connective among $\{\vee, \wedge\}$ and where y and x_i are propositional literals. For a given gate g, we define $CNF(g)$ as the set of clauses encoding g. A propositional variable y (resp. $x_1, \ldots, x_k$) is an *output variable* (resp. are *input variables*) of a gate of the form $y = f(x'_1, \ldots, x'_k)$, where $x'_i \in \{x_i, \neg x_i\}$. Finally, we define a *circuit sat* formula as a conjunction of gates ($\mathcal{G}$) and clauses (C). It is called a full circuit, when $C = \emptyset$.

Our approach is inspired by two related works of Purdom [8] and Boufkhad [2]. In [8], P. Purdom has proposed an original branching criterion (called complementary search) to avoid redundancy during search.

Property 1 (Purdom [8]). Let Σ be a CNF formula, l be a branching literal then Σ is satisfiable iff $\Sigma(l)$ is satisfiable or $\Sigma(\neg l) \wedge \neg\alpha(\neg l)$ is satisfiable.

As noted by Purdom, the exploitation of the property 1 requires additional clauses that can be derived by translating the formula $\neg\alpha(\neg l)$ in Disjunctive Normal Form (DNF) to a CNF formula. This drawback was also noted by Gallo and Urbani [4] :"*Purdom's branching criterion succeeds in reducing the size of the search tree but a price must be paid. In fact, the formula must be transformed into the standard form of set of clauses, which might be quite costly*". For this reason, the property above is only exploited when $\alpha(\neg l)$ is reduced to a single clause (the literal $\neg l$ occurs only once in Σ). The negation of such a clause is a set of unit clauses.

In [2], Boufkhad has defined a concept of restrictive solution. This kind of solution has special characteristics that can be checked in polynomial time and each satisfiable formula has at least one of these special solution. Three variants of these solutions have been proposed : Negative Prime Solution (NPS), Positive Prime Solution (PPS) and Locally Optimized Solution (LOS). Using such restrictive models, Boufkhad *et. al.* obtained a new theoretical upper bound of the threshold of random 3-SAT formula [3]. Similarly to Purdom, another use proposed by Boufkhad [2] is to add new clauses to the formula in order to restrict its set of models to only those with special characteristics.

Definition 1 (NPS, PPS [2]). *An NPS (resp. PPS) is a solution such that variables assigned the value false (resp. true) cannot be individually inverted to* $true$ *(resp.* $false$*) without contradicting the formula.*

Furthermore, Boufkhad [2] introduced the notion of Locally Optimized Solution (LOS) relative to a truth assignment S. It is called optimized in the sense that no better solution can be found by just inverting the value of a variable. It is said locally optimized relative to a truth assignment S because the value assigned to any variable x in S (called the reference value of x in S) is preferred to the opposite one. Any satisfiable formula has at least one LOS relative to any truth assignment S [2].

Property 2 (Boufkhad [2]). Let Σ be a CNF, $S \in \mathcal{L}(\Sigma)$ a consistent set of literals and $C = \bigwedge_{l \in S}(l \vee \neg\alpha(\neg l))$. Σ is satisfiable if and only if $\Sigma \wedge C$ is satisfiable.

From the proof of the property [2], it follows that any solution to $\Sigma \wedge C$ is a LOS relative to S of Σ. Obviously, the two properties 1 and 2 are very similar. For the same reasons as in Purdom, only literals that occur at most twice are considered in [2] leading to additional clauses with at most three literals.

3 Circuit Based Encoding

The results presented in this section can be summarized as follows. First, using auxiliary variables, we avoid the main drawback of Purdom and Boufkhad approaches i.e. the additional constraints can be obtained using linear time approach. Second, the conjunction of the original formula and the additional constraint lead to a circuit SAT formula.

Property 3

1. Let $\Sigma = (l \vee \alpha(l)) \wedge (\neg l \vee \alpha(\neg l)) \wedge \Gamma$ be a CNF. Σ is satisfiable iff $(l = \alpha(\neg l)) \wedge (l \vee \alpha(l)) \wedge \Gamma$ is satisfiable
2. Let $l = \alpha(\neg l) = \wedge(l_1, \ldots, l_m, c_1 \ldots, c_k)$ be a boolean equation, where $|c_i| > 1$. Let y_i be an auxiliary variable representing a clause c_i. The gate $(l = \alpha(\neg l))$ and $\{l = \wedge(l_1, \ldots, l_m, y_1, \ldots, y_k), y_1 = \vee(c_1), \ldots, y_k = \vee(c_k)\}$ are equivalent for SAT.

Property 3.1 illustrates how the added clauses in conjunction with the original ones can be encoded with a boolean gate, whereas, the property 3.2 shows how the gate $l = \alpha(\neg l)$ can be translated in linear time to a set of boolean gates using auxiliary variables.

Example 1. Let $\Sigma = \Gamma \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_4 \vee x_1) \wedge (\neg x_4 \vee x_2) \wedge (\neg x_4 \vee x_3)$ be a CNF formula encoding the gate g ($x_4 = \wedge(x_1, x_2, x_3)$). To illustrate the detection of such explicit gate, we consider two distinct case. First, if $\neg x_4 \notin \mathcal{L}(\Gamma)$, applying the properties 3.1 and 3.2 to literal x_4 we detect the same gate g. In the second case, where $\neg x_4 \in \mathcal{L}(\Gamma)$ i.e. $\{(\neg x_4 \vee \gamma(\neg x_4))\} \subset \Gamma$, we detect a gate g' ($x_4 = \wedge(x_1, x_2, x_3, \ldots)$), the gate g' include the gate g and other auxiliary variables introduced to represent the clauses in $\gamma(\neg x_4)$.

The example 1 shows that when some clauses of the original formula express a gate (explicit gate), our approach can recover such gates in a very simple way. The question of recovering explicit gates has been subject of interesting works by [6,9]. Properties 3.1 and 3.2 describe one step in our encoding. Given a consistent set of literals S, our proposed translation iterates the application of the above properties on each literal of S. The Algorithm 1 describes the encoding of any CNF as a circuit SAT formula. It produces a new formula made of a set of gates and clauses, equivalent w.r.t. SAT to the original formula. Let $uncov(\Sigma, \mathcal{G})$ be the set of clauses of Σ uncovered by $\mathcal{G}$. Condition of while loop avoids multiple clause covering. Second condition in foreach loop avoids the introduction of unnecessary auxiliary variables. Let us remark that the obtained circuit SAT formula is dependent on the ordering of the considered set of literals.

Algorithm 1. CircuitSat(in Σ : CNF, in S : set of literals, out $\mathcal{G}$: set of gates)

begin
  $\mathcal{G} = \emptyset$;
  while $(S \cap \mathcal{L}(uncov(\Sigma, \mathcal{G})) \neq \emptyset)$ **do**
    choose $l \in S$; $In = \emptyset$;
    foreach $c_i \in \alpha(\neg l)$ **do**
      if $|c_i| < 2$ **then** $In = In \cup \{c_i\}$ **else**
        if $(\exists y_j \in \mathcal{G} | y_j = \vee(c_i)$ **then** $In = In \cup \{y_j\}$ **else**
          $\mathcal{G} = \mathcal{G} \cup \{y_i = \vee(c_i)\}$; $In = In \cup \{y_i\}$
      $\mathcal{G} = \mathcal{G} \cup \{l = \wedge(In)\}$; $S = S - \{l\}$;
end

Property 4. Let Σ be a CNF. S is a model of Σ iff the set of gates $\mathcal{G}$ obtained by $CircuitSat(\Sigma, \overline{S})$ is a full circuit encoding i.e. $uncov(\Sigma, \mathcal{G}) = \emptyset$

From the Property 4, we can deduce several interesting results: First, for unsatisfiable formula, our circuit sat encoding can not lead to a full circuit encoding. Second, if a formula is entirely covered using $CircuitSat$, then the formula is satisfiable. Finally, in the general case, finding a full circuit encoding is intractable.

As our proposed algorithm is based on the property 3, if we consider S as a complete assignment of the variables of Σ, it is important to note that any model of the circuit SAT formula is a LOS of Σ with respect to S. Obviously, if we consider $S = \mathcal{L}^+$ (resp. $S = \mathcal{L}^-$), then any model of the circuit formula is an NPS (resp. PPS) of the original formula Σ. In these last two cases, the obtained circuit SAT is described by the following property.

Property 5. Let Σ be a CNF. If $S = \mathcal{L}^+$ (resp. $S = \mathcal{L}^-$) then $CircuitSat(\Sigma, S)$ delivers a set of gates $\mathcal{G}$ such that all clauses of Σ not covered by $\mathcal{G}$ are positive (resp. negative).

In the following example, we show that on structured SAT instances, our algorithm $CircuitSAT(\Sigma, S)$ can deliver interesting new constraints.

Example 2 (Pigeon hole). Let us consider the pigeon hole problem. The problem PH(n) consists in putting all the n pigeons into $n-1$ different holes such that each hole contain at most one pigeon. To encode this problem in CNF formula we need $n \times (n-1)$ propositional variables p_i^j with $i \in \{1,\ldots,n\}$, $j \in \{1,\ldots,n-1\}$. Each variable p_i^j expresses that the pigeon i is in the hole j. The CNF formula $PH(n)$ contains two kind of clauses. (i) $\bigwedge(p_i^1 \vee p_i^2 \ldots p_i^{n-1}), 1 \leq i \leq n$ encoding that the pigeon i is not left free (must be put in a hole) and $\bigwedge(\neg p_i^j \vee \neg p_k^j), 1 \leq j \leq n-1, 1 \leq i < k \leq n$ expressing that two different pigeons (i and k) can not be put in the same hole j. Applying the algorithm 1 using positive literals ($\mathcal{L}^+(\Sigma)$), we obtain $n \times (n-1)$ gates. Each gate g is of the form $p_j^i = \wedge(\neg p_1^i, \neg p_2^i \ldots \neg p_{j-1}^i, \neg p_{j+1}^i, \ldots \neg p_n^i)$ which expresses a new implicit information: "each hole contains exactly one pigeon". On real world problem, we expect that our approach might deduce interesting and meaningful knowledge.

4 Handling Circuit SAT Formula

Circuit SAT and backdoor sets
Circuit SAT formula can be exploited for deriving useful hidden structure of a given problem instance. Following the recent approach proposed in [6] (LSAT) for computing backdoor sets, we show that our circuit SAT representation is suitable for computing such structure. The notion of (strong) Backdoor introduced by Williams *et. al.* in [12] is an active research topic because of its connection to problem hardness. A set of variables forms a backdoor for a given formula if there exists an assignment to these variables such that the simplified formula can be solved in polynomial time. Such a set of variables is called a strong backdoor if any assignment to these variables leads to a tractable sub-formula. This kind of structure is related to the notion of independent variables (see section 2) [10,5].

Since our algorithm is dependent on the chosen set S of literals, one can try to cover in priority clauses that do not belong to the targeted polynomial fragment (e.g. a set of non-horn clauses). Then, our proposed encoding delivers a set of gates and a horn CNF part. Consequently, as unit propagation is complete for horn clauses, only the circuit part is considered in the computation of the strong backdoor.

Circuit SAT based preprocessing
Our circuit encoding delivers an interesting polynomial preprocessing technique. Indeed, the circuit formula, encoding a given instance is more constrained than the original one. Our preprocessing technique is made of two different steps: First, generating a circuit formula using our circuit encoding obtained by processing a set of literals S with a number of occurrences bounded by a given constant k. Then, translating the circuit formula (obtained in the first step) to CNF. Because of space limitation, more details and experimental results are described in a technical report available from the authors [1].

5 Conclusion

We have proposed an original new circuit sat based encoding of CNF formula. It makes an original use of the concept of restrictive models introduced by Boufkhad to

polynomially translate any formula in conjunctive normal form (CNF) to a circuit sat representation (a conjunction of gates and clauses). The derived circuit SAT formula is equivalent with respect to SAT to the original CNF. Our encoding can be used to recover explicit gates and other meaningful structural knowledge and as a preprocessing step to speed up SAT solvers.

References

1. G. Audemard and L. Sais. Circuit Based Encoding of CNF formulas. Technical report, CRIL, 2007.
2. Y. Boufkhad. *Aspects probabilistes et algorithmiques du problème de satisfiabilité*. phd thesis, Université de Paris 6, Laboratoire d' Informatique de Paris 6, 1996.
3. O. Dubois and Y. Boufkhad. A general upper bound for the satisfiability threshold of random r-SAT formulae. *Journal of Algorithms*, 24(2):395–420, August 1997.
4. G. Gallo and G. Urbani. Algorithms for testing the satisfiability of propositional formulae. *journal of logic programming*, 7(1):45–61, July 1989.
5. E. Giunchiglia, M. Maratea, and A. Tacchella. Dependent and independent variables in propositional satisfiability. In *proceedings of JELIA*, volume 2424 of *LNCS*, pages 296–307, 2002.
6. E. Gregoire, B. Mazure, R. Ostrowski, and L. Sais. Automatic extraction of functional dependencies. In *proc. of SAT*, volume 3542 of *LNCS*, pages 122–132,2005.
7. F. Lu, L. Wang, K. Cheng, and R. Huang. A circuit sat solver with signal correlation guided learning. In *proceedings of international conference DATE*, pages 892–897, 2003.
8. P. W. Purdom. Solving satisfiability with less searching. *IEEE transactions on pattern analysis ans machine intelligence*, PAMI-6(4):510–513, July 1984.
9. J Roy, I. Markov, and V. Bertacco. Restoring circuit structure from SAT instances. In *proceedings of international workshop on Logic and synthesis*, 2004.
10. B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *proceedings of IJCAI*, 1997.
11. C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with DPLL search. In *proceedings of international conference CP*, pages 663–678, 2004.
12. R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *proceedings of IJCAI*, pages 1173–1178, 2003.

Breaking Symmetries in SAT Matrix Models

Inês Lynce[1] and Joao Marques-Silva[2]

[1] IST/INESC-ID, Technical University of Lisbon, Portugal
ines@sat.inesc-id.pt
[2] School of Electronics and Computer Science, University of Southampton, UK
jpms@ecs.soton.ac.uk

Abstract. Symmetry occurs naturally in many computational problems. The use of symmetry breaking techniques for solving search problems reduces the search space and therefore is expected to reduce the search time. Recent advances in breaking symmetries in SAT models are mainly focused on the identification of permutable variables via graph automorphism. These symmetries are denoted as instance-dependent, and although shown to be effective for different problem instances, the advantages of their generalised use in SAT are far from clear. Indeed, in many cases symmetry breaking predicates can introduce significant computational overhead, rendering ineffective the use of symmetry breaking. In contrast, in other domains, symmetry breaking is usually achieved by identifying instance-independent symmetries, often with promising experimental results. This paper studies the use of instance-independent symmetry breaking predicates in SAT. A concrete application is considered, and techniques for symmetry breaking in matrix models from CP are used. Our results indicate that instance-independent symmetry breaking predicates for matrix models can be significantly more effective than instance-dependent symmetry breaking predicates.

1 Introduction

In the recent past, symmetry breaking has been proposed as a technique that may be essential for solving hard computational problems. Indeed, successful results have been reported in different areas, including satisfiability (SAT), constraint programming (CP), planning and model checking. Nonetheless, whereas in most areas symmetries are broken according to specific properties of each problem instance, in Boolean satisfiability a more generic approach is often followed [1]. Instead of breaking symmetries when modelling a problem instance with SAT, generic symmetry breaking tools read a CNF formula and output the given formula extended with symmetry breaking clauses, which result from a graph automorphism analysis.

State-of-the-art SAT solvers are currently able to deal with very large formulae and to perform hundreds of thousands of propagations per second. Hence, one may think that augmenting the formula with symmetry breaking clauses in a preprocessing step does not represent a significant overhead to a SAT solver.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 22–27, 2007.

However, this is not the case for preprocessing techniques in general. Only specific techniques applied to specific problems have been shown to be effective.

On the other hand, mainly due to the effectiveness of SAT solvers learning techniques, modelling has not been much developed in SAT, at least when compared with other areas such as CP. Jointly with dynamic heuristics, learning is able to extend the formula in such a way that strategic resolution steps are performed. So, it is a reasonable approach to let the SAT solver learn *intelligently* rather than telling in advance what it should be able to learn during search. However, learning can hardly replace symmetry breaking predicates. For example, symmetry breaking predicates may reduce the number of solutions and learning does not. This paper compares the use of generalised CNF-based symmetry breaking predicates, also known as *instance-dependent predicates*, with the use of specific symmetry breaking predicates, i.e. *instance-independent predicates*, in the context of SAT matrix models[1].

2 Symmetry Breaking in SAT

The first complete framework suggesting a symmetry extraction mechanism for satisfiability based on a reduction to graph automorphism was proposed in [2]. This approach has been recently adapted and made practical for satisfiability in `shatter` [1]. For single variable permutations, `shatter` generates CNF formulae linear in the number of variables. In addition, `shatter` proposes a number of optimisations to the implementation of the graph automorphism algorithm. (Observe, however, that graph automorphism is believed not to be in P, even though it is not known whether it is NP-complete.)

The same authors have compared the efficiency of breaking *instance-dependent* symmetries against the efficiency of breaking *instance-independent* symmetries [7]. For the concrete problem of exact graph colouring, the use of *instance-dependent* symmetries is significantly more efficient. Instance-dependent symmetries are identified *automatically* via graph automorphism, whereas instance-independent symmetries are specific to the problem and are usually identified *manually* at the time the encoding is done. Before the existence of an efficient tool such as `shatter`, the generation of effective instance-independent symmetries was studied for several classes of combinatorial objects [8]. However, this approach was not evaluated against a generic one. Moreover, the use of symmetry breaking predicates in local search consistently has a negative effect in local search algorithms [6]. Interestingly, this observation has motivated an opposite strategy when applying local search: *maximising* symmetry in the SAT model.

3 Symmetry Breaking in Matrix Models

Symmetry in matrix models is usually broken by using lexicographic constraints [3]. If permutations in rows and/or columns can be made without affecting the

[1] The paper follows the classification of predicates proposed in [1].

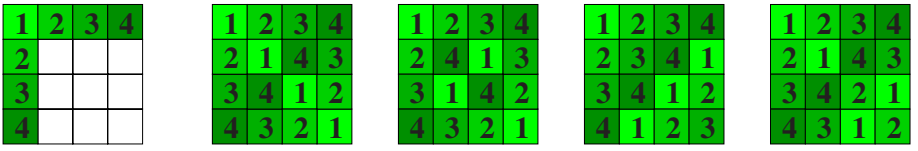


Fig. 1. A 4x4 Latin square with the first row and column fixed and its 4 solutions

existence of solutions, then an ordering should be fixed to eliminate these symmetries. Although different orderings may be used, lexicographic ordering is considered to be the most intuitive. The resulting predicates are not guaranteed to eliminate all symmetries, since the problem instance may contain other symmetries. Also, ordering constraints do not break all symmetries when matrices have both row and column symmetries [3]. Nevertheless, symmetries in matrix models have the advantages of being easily identified and broken at a small cost.

Example 1. Consider a 4x4 Latin square, i.e. a 4x4 matrix to be filled with 4 different symbols in such a way that each symbol occurs exactly once in each row and exactly once in each column. This problem has 576 solutions. Clearly, most symmetries can be easily eliminated by forcing the first row and the first column to be lexicographically ordered. Nonetheless, these constraints do not prevent this problem from having more than one solution: there are still 4 possible solutions. Figure 1 illustrates a 4x4 Latin square after adding the lexicographic constraints and the four possible solutions. `Shatter` is able to identify further symmetries such that only two of these solutions can be found.

We now focus on the SHIPs SAT model [4,5]. SHIPs is a SAT-based approach for solving the problem of haplotype inference by pure parsimony (HIPP). (A detailed description of SHIPs can be found in [4,5].) Given a set $\mathcal{G}$ of n genotypes, each of length m, the haplotype inference problem consists in finding a set $\mathcal{H}$ of $2 \cdot n$ haplotypes, not necessarily different, such that for each genotype $g_i \in \mathcal{G}$ there is at least one pair of haplotypes (h_j, h_k), with h_j and $h_k \in \mathcal{H}$ such that the pair (h_j, h_k) explains g_i. The pure parsimony approach finds a solution that minimises the total number of distinct haplotypes used.

The organisation of the SHIPs algorithm considers increasing values r of candidate haplotypes, with $1 \leq r \leq 2 \cdot n$, such that a solution is found when r haplotypes suffice to explain the n genotypes. The SHIPs model [4,5] can be described by the matrix formulation $G = S^a \cdot H \oplus S^b \cdot H$, where G is a $n \times m$ matrix describing the genotypes, H is a $r \times m$ matrix of haplotype variables, S^a and S^b are $n \times r$ matrices of selector variables, and $\oplus$ is the explanation operation. One of the contributions of the SHIPs model are the techniques for breaking key symmetries in the problem formulation. If matrix H is interpreted as a vector of strings of size m, $H = [h_1 h_2 \ldots h_r]^T$, then we can impose the condition $h_1 < h_2 < \ldots < h_r$, i.e. the haplotypes are lexicographically sorted.An additional form of symmetry is due to the S variables. If $S^a = [s_1^a \ldots s_n^a]^T$ and $S^b = [s_1^b \ldots s_n^b]^T$, then we can impose the condition $s_i^a \leq s_i^b$, $1 \leq i \leq n$, i.e. for

each genotype i, the strings representing the selector variables a and the selector variables b are lexicographically ordered.

4 Experimental Results

This section provides empirical evidence that breaking instance-independent symmetry in SAT matrix models can be more effective than breaking instance-dependent symmetries. Different encodings for the SHIPs matrix model are evaluated. Also, due to the incremental approach implemented in SHIPs, both satisfiable and unsatisfiable problem instances are obtained. Consider a solution with size s: then iterations with $r < s$ represent unsatisfiable instances, and the iteration with $r = s$ represents a satisfiable instance. A set of 1183 problem instances obtained from `http://www.stats.ox.ac.uk/~marchini/phaseoff.html` and from [5] were evaluated. The results were obtained on an Intel Xeon 5160 (3.0GHz with 4GB of RAM) and a timeout of 1000s.

From an initial universe of 1183 instances, we removed 348 instances with equal computed lower and upper bounds [4]. Of the remaining instances, 134 are aborted when symmetry breaking is not used and 74 are aborted when symmetry breaking is used. Moreover, the run times with symmetry breaking are also consistently smaller. Figure 2 provides two plots comparing the effect of breaking instance-independent symmetries in terms of the total CPU time for unsatisfiable and satisfiable instances, respectively. Clearly, for unsatisfiable instances it is *always* useful to break symmetries, whereas for satisfiable instances it is useful in most cases. Next, we compare the use of `shatter` [1] on each set of unsatisfiable and satisfiable instances. `Shatter` may be applied either to the CNF formula resulting from the SHIPs model, for which instance-independent symmetry breaking predicates have been included, or to the plain model, for which no symmetries are broken. Figure 3 compares both approaches. Even

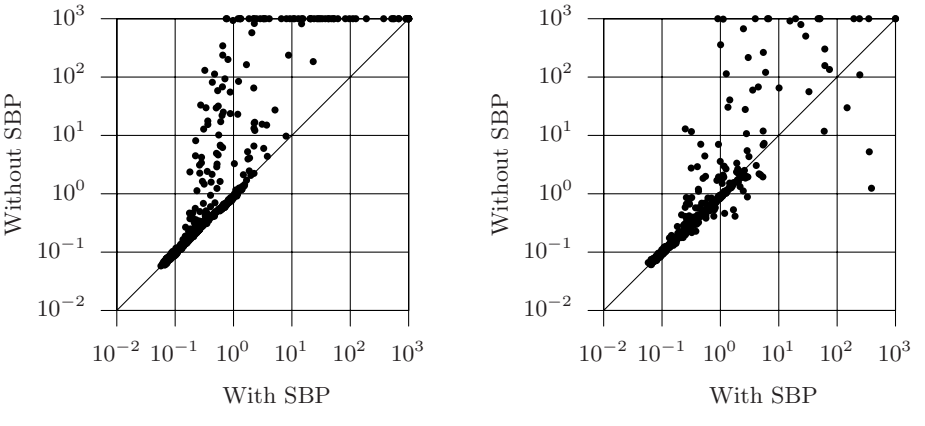


Fig. 2. CPU times with and without instance-independent symmetry breaking predicates (SBP) on unsatisfiable and satisfiable instances

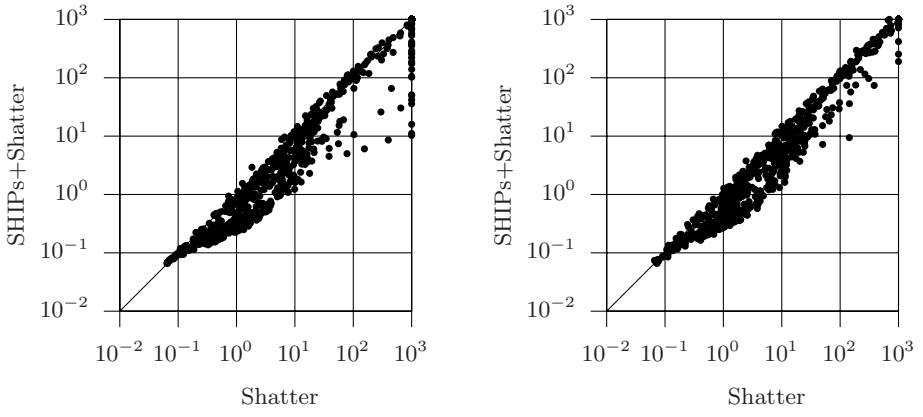


Fig. 3. Shatter vs SHIPS+Shatter on unsatisfiable and satisfiable instances

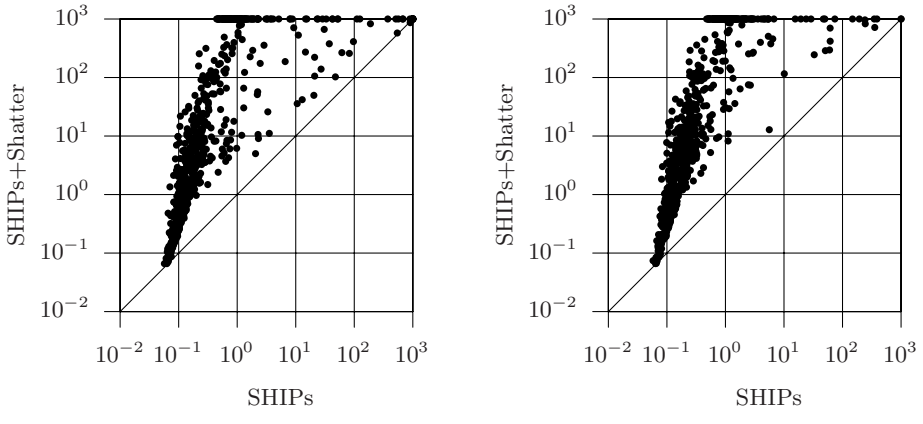


Fig. 4. SHIPs vs SHIPS+Shatter on unsatisfiable and satisfiable instances

though `shatter` performs better on the SHIPs model rather than on the plain model, the difference is not significant, in particular for satisfiable instances.

Finally, Figure 4 compares the use of instance-independent symmetry breaking predicates (i.e. SHIPs) with the use of both instance-independent and instance-dependent symmetry breaking predicates (i.e. SHIPs+Shatter) in terms of CPU time. The use of instance-independent symmetry breaking predicates is consistently more efficient than the use of both types of symmetry breaking predicates. Moreover, `Shatter` in unable to break all the symmetries in the allowed CPU time (1000s) for many instances. This is probably due to these instances having many symmetries, which can be easily identified beforehand.

One additional question is: "If there was an oracle giving the CNF formula computed by `shatter` what would be the SAT solver performance?" With this purpose, the formula computed by `shatter` within 1000s was given to the SAT

solver. Then we compared the time required by SHIPs with the time required by the SAT solver on the formula computed by `shatter`. If `shatter` is run on the plain model, i.e. without symmetry breaking predicates, then the SAT solver is able to solve more problem instances than using the plain model, but still less 45 instances than SHIPs. Also, the instances not solved by SHIPs are also not solved after using `shatter`. If `shatter` is run on the SHIPs model, which includes symmetry breaking predicates, then exactly the same instances are not solved. For the instances solved, the use of `shatter` yields a negligible speedup.

5 Conclusions and Future Work

Despite its impact in CP, symmetry breaking is seldom used in SAT. The main reason is that symmetry breaking can be time-consuming and not always effective in modern SAT solvers. This paper explores a different line of research, which has been quite successful in CP: instead of considering instance-dependent symmetry breaking, we propose problem-specific instance-independent symmetry breaking. Clearly, this necessarily depends on the application domain. The paper focus on symmetry breaking techniques for SAT matrix models, and more concretely for the HIPP problem. The experimental results show that more careful modelling of computational problems with SAT techniques, and exploring well-established symmetry breaking techniques, can be a quite effective approach, and can significantly outperform existing instance-dependent symmetry breaking approaches.

References

1. F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Solving difficult instances in the presence of symmetry. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1117–1137, 2003.
2. J. M. Crawford, M. L. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 1996.
3. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *International Conference on Principles and Practice of Constraint Programming (CP)*, 2002.
4. I. Lynce and J. Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *National Conference on Artificial Intelligence (AAAI)*, 2006.
5. I. Lynce and J. Marques-Silva. SAT in bioinformatics: Making the case with haplotype inference. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2006.
6. S. Prestwich. First-solution search with symmetry breaking and implied constraints. In *CP Workshop on Modelling and Problem Formulation*, 2001.
7. A. Ramani, I. L. Markov, K. A. Sakallah, and F. A. Aloul. Breaking instance-independent symmetries in exact graph coloring. *Journal of Artificial Intelligence Research*, 26:289–322, 2006.
8. I. Shlyakhter. Generating effective symmetry-breaking predicates for search problems. In *LICS Workshop on Theory and Applications of Satisfiability Testing*, 2001.

Partial Max-SAT Solvers with Clause Learning*

Josep Argelich and Felip Manyà

Computer Science Department
Universitat de Lleida
Jaume II, 69, E-25001 Lleida, Spain
{jargelich,felip}@diei.udl.es

Abstract. We describe three original exact solvers for Partial Max-SAT: PMS, PMS-hard, and PMS-learning. PMS is a branch and bound solver which incorporates efficient data structures, a dynamic variable selection heuristic, inference rules which exploit the fact that some clauses are hard, and a good quality lower bound based on unit propagation. PMS-hard is built on top of PMS and incorporates clause learning only for hard clauses; this learning is similar to the learning incorporated into modern SAT solvers. PMS-learning is built on top of PMS-hard and incorporates learning on both hard and soft clauses; the learning on soft clauses is quite different from the learning on SAT since it has to use Max-SAT resolution instead of SAT resolution. Finally, we report on the experimental investigation in which we compare the state-of-the-art solvers Toolbar and ChaffBS with PMS, PMS-hard, and PMS-learning. The results obtained provide empirical evidence that Partial Max-SAT is a suitable formalism for representing and solving over-constrained problems, and that the learning techniques we have defined in this paper can give rise to substantial performance improvements.

1 Introduction

In recent years we have seen an increasing interest in designing and implementing Max-SAT solvers [1,11,15,16,17,20,21], as well as on studying inference systems for Max-SAT [6,7,15]. Significant progress has been made, and state-of-the-art Max-SAT solvers are able to solve a large number of instances that were beyond the reach of the solvers developed just five years ago. As a proof of the interest in Max-SAT, we highlight the First Max-SAT Evaluation which was held as a colocated event of SAT-2006.

In this paper we focus on Partial Max-SAT, which is a problem between SAT and Max-SAT which is more well-suited for representing and solving over-constrained problems, and has not received yet so much attention by our scientific community. A Partial Max-SAT instance is a CNF formula in which some clauses are *relaxable* or *soft* and the rest are *non-relaxable* or *hard*. Solving a Partial Max-SAT instance amounts to find an assignment that satisfies all the hard clauses and the maximum number of soft clauses.

* Research partially supported by projects TIN2004-07933-C03-03 and TIN2006-15662-C02-02 funded by the *Ministerio de Educación y Ciencia*.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 28–40, 2007.

Let us illustrate with an example the expressive power of Partial Max-SAT. Assume we want to solve the problem of coloring a graph with two colors in such a way that the minimum number of adjacent vertices are colored with the same color. If we consider the graph with vertices $\{v_1, v_2, v_3\}$ and with edges $\{(v_1, v_2), (v_1, v_3), (v_2, v_3)\}$, that problem is encoded as a Partial Max-SAT instance as follows: (i) the set of propositional variables is $\{v_1^1, v_1^2, v_2^1, v_2^2, v_3^1, v_3^2\}$; the intended meaning of variable v_i^j is that vertex v_i is colored with color j; (ii) the hard clauses are the following at-least-one and at-most-one clauses:

$$[v_1^1 \vee v_1^2], [\neg v_1^1 \vee \neg v_1^2], [v_2^1 \vee v_2^2], [\neg v_2^1 \vee \neg v_2^2], [v_3^1 \vee v_3^2], [\neg v_3^1 \vee \neg v_3^2];$$

and (iii) there are the following soft clauses:

$$(\neg v_1^1 \vee \neg v_2^1), (\neg v_1^2 \vee \neg v_2^2), (\neg v_1^1 \vee \neg v_3^1), (\neg v_1^2 \vee \neg v_3^2), (\neg v_2^1 \vee \neg v_3^1), (\neg v_2^2 \vee \neg v_3^2).$$

Note that we write hard clauses between square brackets in order to distinguish hard clauses from soft clauses.

In this paper we describe three original exact solvers for Partial Max-SAT: PMS, PMS-hard, and PMS-learning. PMS is a branch and bound solver which incorporates efficient data structures, a dynamic variable selection heuristic, inference rules which exploit the fact that some clauses are hard, and a good quality lower bound based on unit propagation. PMS-hard is built on top of PMS and incorporates clause learning only for hard clauses; this learning is similar to the learning incorporated into modern SAT solvers. PMS-learning is built on top of PMS-hard and incorporates learning on both hard and soft clauses; the learning on soft clauses is quite different from the learning on SAT since it has to use Max-SAT resolution instead of SAT resolution. SAT resolution preserves satisfiability but does not preserve the number of unsatisfied clauses as in Max-SAT resolution. Finally, we report on the experimental investigation in which we compare the state-of-the-art solvers Toolbar [11,15] and ChaffBS [9] with PMS, PMS-hard, and PMS-learning. The results obtained provide empirical evidence that Partial Max-SAT is a suitable formalism for representing and solving over-constrained problems, and that the learning techniques we have defined in this paper can give rise to substantial performance improvements.

It is worth to point out that, to the best of our knowledge, learning of soft clauses has not been defined and incorporated into any Max-SAT or Partial Max-SAT solver. The learning mechanism of soft clauses defined in this paper can be applied to Max-SAT solvers too.

The paper is structured as follows. In Section 2 we present the most relevant related work. In Section 3 we describe in detail PMS, PMS-hard and PMS-learning. In Section 4 we report on the experimental investigation. Finally, we present some concluding remarks.

2 Previous Work

The first local search solvers for Partial Max-SAT were defined by Jiang et al. [14] and Cha et al. [8]. In SAT-2006, Fu and Malik [9] presented two exact

Partial Max-SAT algorithms which use the SAT solver zChaff to solve Partial Max-SAT: the first algorithm, which is diagnosis based, iteratively analyzes the UNSAT core of the current SAT instance and eliminates the core through a modification of the problem instance by adding relaxation variables. The second algorithm, which is encoding based, constructs an efficient auxiliary counter that constraints the number of relaxed clauses and supports binary search or linear scan for the optimal solution.

Argelich and Manyà [2,3] defined branch and bound solvers for solving the Partial Max-SAT problem for a formalism, called soft CNF formulas, in which blocks of soft clauses are considered instead of individual clauses. Their solvers do not incorporate any clause learning technique.

Argelich and Manyà [4] defined the first learning scheme of hard clauses for a specialized partial Max-SAT solvers. This learning scheme is the learning of hard clauses described in the present paper.

3 Partial Max-SAT Solvers

We first define a basic branch and bound Partial Max-SAT solver and then explain the main features we added to the basic solver in order to obtain PMS, PMS-hard and PMS-learning.

The space of all possible assignments for a Partial Max-SAT instance ϕ can be represented as a search tree, where internal nodes represent partial assignments and leaf nodes represent complete assignments. A branch and bound (BnB) algorithm explores that search tree in a depth-first manner. At each node, the algorithm backtracks if the current partial assignment violates some hard clause, and applies the one-literal rule [18] to the literals that occur in unit hard clauses; i.e., given a literal $\neg p$ (p), it deletes all the clauses containing the literal $\neg p$ (p) and removes all the occurrences of the literal p ($\neg p$). If the current partial assignment does not violate any hard clause, the algorithm compares the number of soft clauses unsatisfied by the best complete assignment found so far, called upper bound (ub), with the number of soft clauses unsatisfied by the current partial assignment, called lower bound (lb). Obviously, if $ub \leq lb$, a better assignment cannot be found from this point in search. In that case, the algorithm prunes the subtree below the current node and backtracks to a higher level in the search tree. If $ub > lb$, it extends the current partial assignment by instantiating one more variable, say p, which is selected using the following heuristic: it instantiates first the variables that appear most often; ties are broken using the lexicographical order. The instantiation of p leads to the creation of two branches from the current branch: the left branch corresponds to instantiating p to false, and the right branch corresponds to instantiating p to true. In that case, the formula associated with the left (right) branch is obtained from the formula of the current node by applying the one-literal rule using the literal $\neg p$ (p). The value that ub takes after exploring the entire search tree is the minimum number of soft clauses that cannot be satisfied by a complete assignment that satisfies all the hard clauses.

In contrast to Max-SAT solvers, Partial Max-SAT solvers enforce unit propagation on unit hard clauses. This is not possible in Max-SAT because unit propagation on soft clauses is unsound; i.e., the number of clauses in the original formula unsatisfied by any assignment can be different from the number of unsatisfied clauses in the simplified formula obtained by applying unit propagation. Moreover, a branch of the search tree can be pruned as soon as a hard clause is violated, independently if the lower bound has reached the upper bound.

3.1 PMS

PMS adds to the previous algorithm the following features:

- Variable selection heuristic: it uses the two-sided Jeroslow-Wang rule [13].
- Lower bound: it implements a variant of lower bound UP [16,17] adapted to Partial Max-SAT. In lower bound UP for Max-SAT, the lower bound is the current number of unsatisfied clauses plus an underestimation of the minimum number of clauses that will become unsatisfied if the current partial assignment is extended to a complete assignment. Such an underestimation is the number of disjoint unsatisfiable subsets that can be detected using unit propagation.
  In lower bound UP for Partial Max-SAT, the underestimation is the number of unsatisfiable subsets that can be derived by applying unit propagation in such a way that soft clauses appear only in one subset. In UP for Max-SAT, the clauses in unsatisfiable subsets can appear just in one subset. In Partial Max-SAT, hard clauses can appear in more than one subset. This is a crucial point for obtaining a better performance profile than in Max-SAT for some instances. We implement UP using the data structure formed by two queues as described in [17].
- The initial upper bound is computed with a local search solver.
- Inference rules: the algorithm applies the complementary unit clause rule: it replaces two complementary unit clauses with an empty clause. It also applies the almost common clause rule [5] as a preprocessing: it replaces any two clauses of the form $x \vee y, \neg x \vee y$ with y. In both cases, if some of the premises is hard, it is not removed.

3.2 PMS-Hard

PMS-hard extends PMS with a learning module that analyzes the conflicts detected in hard clauses. When a conflict is detected, it analyzes the conflicting clause detected using the 1-UIP learning scheme [19] implemented in zChaff [22], and learns a hard clause. The mission of the conflict clauses added is to avoid visiting regions of the search space that cannot lead to an optimal solution due to fact that some hard clause is violated.

Since any optimal solution to a Partial Max-SAT instance satisfies all the hard clauses, the fact of adding redundant clauses does not affect the number of unsatisfied soft clauses. So, we can guarantee that the number of unsatisfied clauses is preserved by our clause learning module.

As we will see in the experimental investigation, that learning scheme produces significant performance improvements. We introduced first this learning scheme in [4]. It was, to the best of our knowledge, the first time that learning was incorporated into a a branch and bound Partial Max-SAT solver.

3.3 PMS-Learning

PMS-learning extends PMS-hard with a module that analyzes the conflicts detected in which at least one of the conflict clauses is soft. For the time being, our soft learning consist of applying Max-SAT resolution to two conflict clauses. These clauses are selected as follows: between all the conflict clauses, we choose the pairs of clauses $x \vee A$ and $\neg x \vee B$ that have the minimum number of literals and, finally, we choose the pair that has the minimum number of different literals among A and B. We give priority to resolve a hard clause and a soft clause.

The Max-SAT resolution rule [6,11,15] corresponds to Rule 1 of Figure 1. The application of the rule consists of replacing the premises with the conclusions; this way the number of unsatisfied clauses is preserved. Moreover, it has been shown that the rule provides a complete calculus for Max-SAT [6,7].

In the case of partial Max-SAT, Max-SAT resolution can be simplified when at least one of the premises is hard by applying the next rule, which is called absorption rule in [15]:

$$\frac{\begin{array}{l}[D] \\ D \vee D'\end{array}}{[D]}$$

where D and D' are disjunctions of literals.

Rule 1

$$\frac{\begin{array}{l}x \vee a_1 \vee \cdots \vee a_s \\ \overline{x} \vee b_1 \vee \cdots \vee b_t\end{array}}{\begin{array}{l}a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\ x \vee a_1 \vee \cdots \vee a_s \vee \overline{b_1} \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \overline{b_2} \\ \cdots \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t} \\ \overline{x} \vee b_1 \vee \cdots \vee b_t \vee \overline{a_1} \\ \overline{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \overline{a_2} \\ \cdots \\ \overline{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \cdots \vee a_{s-1} \vee \overline{a_s}\end{array}}$$

Rule 2

$$\frac{\begin{array}{l}x \vee a_1 \vee \cdots \vee a_s \\ [\overline{x} \vee b_1 \vee \cdots \vee b_t]\end{array}}{\begin{array}{l}[\overline{x} \vee b_1 \vee \cdots \vee b_t] \\ a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\ x \vee a_1 \vee \cdots \vee a_s \vee \overline{b_1} \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \overline{b_2} \\ \cdots \\ x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t}\end{array}}$$

Rule 3

$$\frac{\begin{array}{l}[x \vee a_1 \vee \cdots \vee a_s] \\ [\overline{x} \vee b_1 \vee \cdots \vee b_t]\end{array}}{\begin{array}{l}[x \vee a_1 \vee \cdots \vee a_s] \\ [\overline{x} \vee b_1 \vee \cdots \vee b_t] \\ [a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t]\end{array}}$$

Fig. 1. Resolution for Partial Max-SAT

The calculus formed by Max-SAT resolution rule and the absorption rule is complete for partial Max-SAT. This follows from the fact that Max-SAT resolution is complete for Max-SAT and the absorption rule is sound (i.e., it preserves the number of unsatisfied clauses). An alternative way of expressing this calculus is by means of Rule 1, Rule 2, and Rule 3 in Figure 1. We follow this approach because it is easy to understand in our context.

Actually, our soft learning mechanism applies Rule 1 when both conflict clauses are soft, and Rule 2 when one conflict clause is hard and the other is soft. When both conflict clauses are hard, it applies the 1-UIP learning scheme [19].

Another learning scheme that could be implemented consists of learning a clause c for every conflict detected in every failed branch, where c is a reason of the conflict. When the lower bound is greater than or equal to the upper bound, we learn one clause for every conflict. Then, we add as a hard clause the disjunction of all the clauses learned in the branch. The main drawback of this approach is that the learned clauses are too big when the minimum number of unsatisfied clauses is not small.

4 Experimental Investigation

We next report the experimental investigation we conducted to compare our solvers (PMS, PMS-hard and PMS-learning) with the following solvers:

- Toolbar: it is the best performing Partial Max-SAT solver according to the results of the Max-SAT Evaluation 2006. We used the last version of Toolbar (version 3.1) [1], which exploits the fact of having hard and soft clauses. We used the default parameters.
  It is worth to mention that even when Toolbar is typically presented as a weighted Max-SAT solver, it is actually a weighted Partial Max-SAT solver.
- ChaffBS: it is a Partial Max-SAT solver implemented on top of the SAT solver zChaff that was presented at SAT-2006 [9]. It solves Partial Max-SAT by encoding it into SAT. This solver is used on the more structured instances, where it has a good performance profile. We do not give experimental results for random Partial Max-2-SAT, random Partial Max-3-SAT and random 2-SoftSAT because it is not competitive for these problems.

On some experiments we also give results with PMS-noLB, which is PMS-hard without computing any underestimation in the lower bound.

All the experiments were performed on a Linux Cluster where the nodes have a 2GHz AMD Opteron processor with 1Gb of RAM.

We used four sets of benchmarks:

- Random Partial Max-2-SAT instances with a number of clauses ranging from 1000 to 3000 and with 100 variables, and Partial Max-3-SAT instances with a number of clauses ranging from 200 to 700 and with 100 variables. These are typical random 2-SAT/3-SAT instances in which 100 clauses are declared, at random, as hard and the rest are declared as soft.

[1] Available at http://mulcyber.toulouse.inra.fr/projects/toolbar/

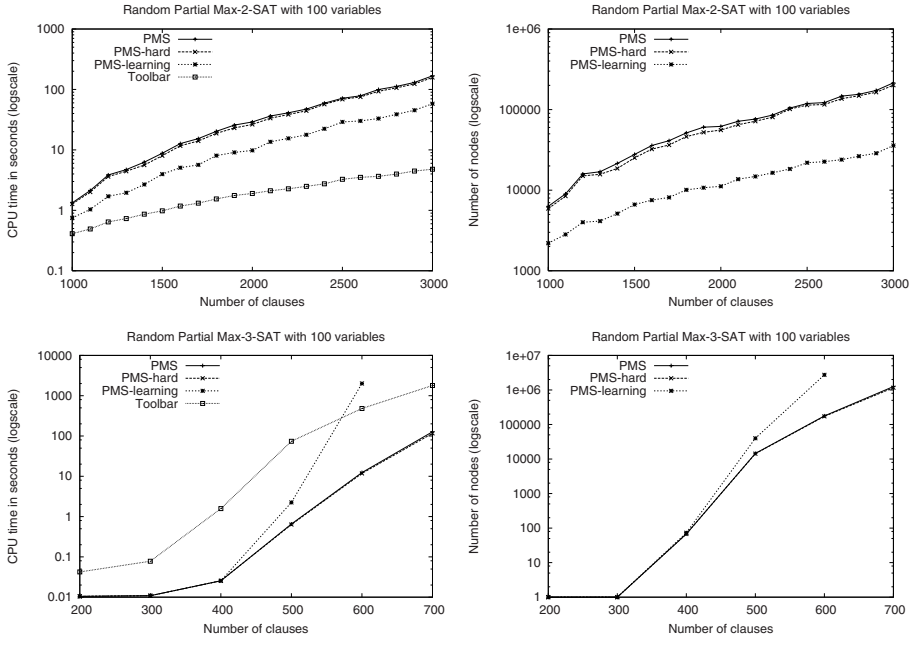


Fig. 2. Random Partial Max-2-SAT and Max-3-SAT instances

- Random 2-SoftSAT instances generated with the algorithm described in [10]. These instances are harder than random Partial Max-2-SAT instances. We solved instances with 150 variables and 150 hard clauses varying the density from 5 to 15. By density we mean the ratio of number of clauses to number of variables.
- Benchmarks from the SAT-2002 Competition [2]. We used benchmarks from the SAT-2002 Competition because they are not so hard as the benchmarks from subsequent competitions. These are satisfiable instances to which we solved the Max-One problem (i.e., compute the maximum number of variables that can be assigned to true by any satisfying assignment).
- Weighted Max-SAT instances from the Max-SAT Evaluation 2006, submitted by the developers of Toolbar, which are Partial Max-SAT instances [3].

The results of solving random Partial Max-2-SAT and random Partial Max-3-SAT instances are shown in Figure 2. We solved 100 instances for each data point. The left plots display the mean time needed to solve an instance with

[2] http://www.satlib.org/Benchmarks/SAT/New/Competition-02/sat-2002-beta.tgz

[3] We mean that they are weighted Max-SAT instances in which some clauses have weight 1 and the rest of clauses have a weight which is bigger than the total number of clauses. These weighted instances are equivalent to Partial Max-SAT instances in which clauses with weight 1 are declared to be soft and the rest of clauses are declared to be hard.

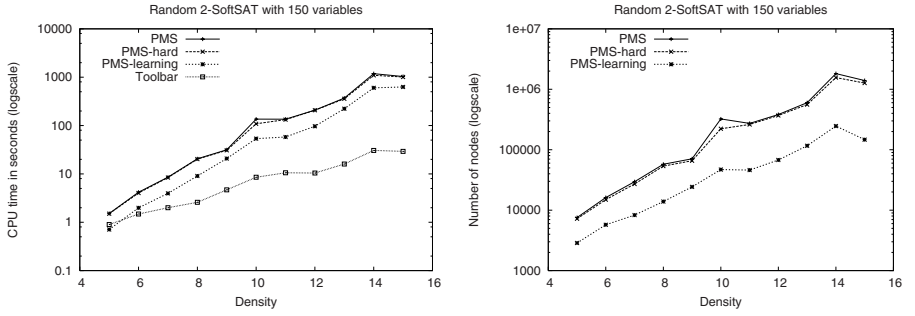


Fig. 3. Random 2-SoftSAT instances

PMS, PMS-hard, PMS-learning and Toolbar. The right plots display the mean number of nodes traversed by our solvers. We observe that the best performing solver for Partial Max-2-SAT is Toolbar while the best performing solvers for Partial Max-3-SAT are PMS and PMS-hard. We believe that the good behavior of Toolbar on Partial Max-2-SAT is due to the incorporation of several Max-SAT inference rules which perform very well for binary clauses. In this example, it is particularly interesting to observe the performance improvements achieved on Partial Max-2-SAT by incorporating our learning scheme of soft clauses.

The results of solving random 2-SoftSAT instances are shown in Figure 3. We solved 100 instances for each data point. The left plot displays the mean time needed to solve an instance with PMS, PMS-hard, PMS-learning and Toolbar. The right plot displays the mean number of nodes traversed by our solvers. We observe that the best performing solver is Toolbar, and that when we apply soft learning we get important gains both in time and in number of nodes. The gains in number of nodes are superior to the gains in time due to the overhead of applying learning.

The results of solving the benchmarks from the SAT-2002 Competition, using a cutoff of 3600 seconds, are shown in Table 1 and Table 2. The first column of Table 1 shows the name of the set of instances, the second column shows the number of instances in the set, the rest of columns show the median time

Table 1. Benchmarks from the SAT-2002 Competition solving the Max-One problem. Time in seconds.

Instance set	#	Toolbar	ChaffBS	PMS-noLB	PMS	PMS-hard	PMS-learning
3-coloring	30	662.95(20)	**3.77(30)**	31.45(30)	818.62(19)	241.74(30)	53.54(20)
AIM	12	317.28(8)	**0.25(12)**	0.35(12)	91.98(10)	0.41(12)	0.37 (12)
CNT	6	0.00(0)	**143.11(4)**	30.87(3)	86.05(1)	155.26(2)	137.96(2)
DP	11	1035.65(1)	411.68(4)	**131.79(6)**	594.71(3)	598.21(4)	638.60(4)
EZFACT	10	0.00(0)	**2.56(10)**	9.11(10)	2739.14(1)	214.10(10)	69.26(8)
MED	4	0.00(0)	52.72(1)	**1.93(1)**	4.25(1)	4.10(1)	6.72 (1)

Table 2. Benchmarks from the SAT-2002 Competition solving the Max-One problem. Number of nodes.

Instance set	#	PMS-noLB	PMS	PMS-hard	PMS-learning
3-coloring	30	429157(30)	2714206(19)	**425872(30)**	93769(20)
AIM	12	2988(12)	3414797(10)	**1842(12)**	1929(12)
CNT	6	**519660(3)**	650615(1)	137756(2)	119154(2)
DP	11	**4513674(6)**	99533(3)	140412(4)	178220(4)
EZFACT	10	**395677(10)**	5774450(1)	**395677(10)**	119923(8)
MED	4	56050(1)	32278(1)	28881(1)	**16236(1)**

(among the instances solved within the cutoff) needed to solve an instance, and the number of instances solved (in brackets). Table 2 is like Table 1 but shows number of nodes instead of time for PMS-noLB, PMS, PMS-hard and PMS-learning. We observe that the best performing solver is ChaffBS and then PMS-noLB. The underestimation of the lower bound seem not to be very useful for these Max-One instances. Among PMS, PMS-hard and PMS-learning, the best option is PMS-hard.

Figure 4 displays the number of instances x from the SAT-2002 Competition that can be solved in y seconds. In this case, the best solvers are PMS-noLB and ChaffBS, and then PMS-hard and PMS-learning.

The results of solving the benchmarks from the Max-SAT 2006 Evaluation, using a cutoff of 3600 seconds, are shown in Table 3 and Table 4. The best performing solvers are ChaffBS, PMS-hard and PMS-learning. We observe that PMS-learning gives rise to substantial gains in number of nodes.

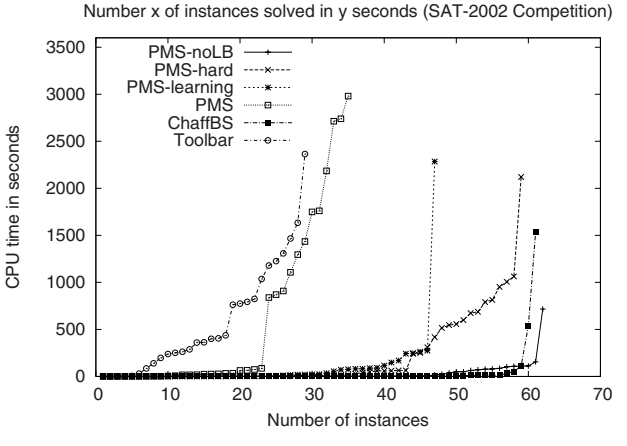


Fig. 4. Number of instances x that can be solved in y seconds. Instances from the SAT-2002 Competition.

Table 3. Benchmarks from the Max-SAT 2006 Evaluation. Time in seconds.

Instance set	#	Toolbar	ChaffBS	PMS	PMS-hard	PMS-learning
Max-Clique	62	**368.18(30)**	294.36(19)	277.94(28)	363.13(29)	412.22(13)
Max-One (3-SAT)	45	237.49(45)	354.04(15)	**1.88(45)**	2.28 (45)	31.00(31)
Max-CSP (Dense Loose)	40	377.42(37)	600.41(36)	1.66 (40)	1.64 (40)	**0.69 (40)**
Max-CSP (Dense Tight)	60	46.95(30)	**239.64(60)**	103.65(50)	101.69(50)	145.98(50)
Max-CSP (Sparse Loose)	40	257.09(38)	20.54(40)	0.39 (40)	0.39 (40)	**0.33 (40)**
Max-CSP (Sparse Tight)	40	236.31(20)	596.19(38)	111.68(40)	**109.70(40)**	112.07(40)
WCSP (N Queens)	7	469.82(6)	**13.96(7)**	18.37(7)	15.17(7)	2.51 (6)

Table 4. Benchmarks from the Max-SAT 2006 Evaluation. Number of nodes.

Instance set	#	PMS	PMS-hard	PMS-learning
Max-Clique	62	3407445(28)	**3492585(29)**	591131(13)
Max-One (3-SAT)	45	32570(45)	**27209(45)**	60028(31)
Max-CSP (Dense Loose)	40	59404(40)	59404(40)	**9736(40)**
Max-CSP (Dense Tight)	60	896789(50)	896789(50)	**283525(50)**
Max-CSP (Sparse Loose)	40	6554(40)	6554(40)	**3060(40)**
Max-CSP (Sparse Tight)	40	385894(40)	385894(40)	**266831(40)**
WCSP (N Queens)	7	239005(7)	**199738(7)**	43761(6)

Figure 5 displays the number of instances x from the Max-SAT 2006 Evaluation that can be solved in y seconds. In this case, the best solvers are PMS-hard and PMS, and then PMS-learning and ChaffBS.

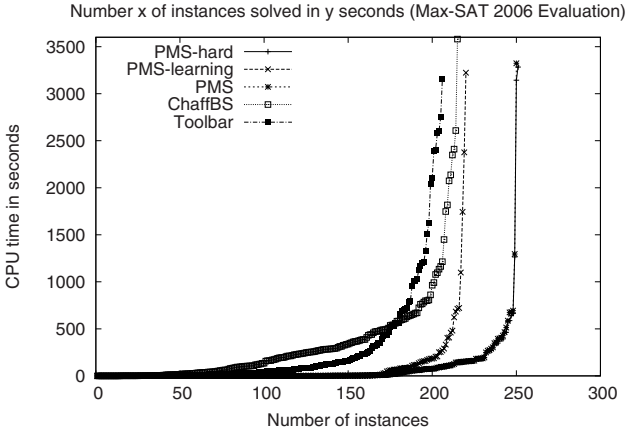


Fig. 5. Number of instances x that can be solved in y seconds. Instances from the Max-SAT Evaluation 2006.

5 Concluding Remarks

In this paper we have designed and implemented three new Partial Max-SAT solvers, and provided empirical evidence that they are competitive. These solvers exploit the fact of knowing which clauses are declared hard and which clauses are declared soft, and incorporate conflict clause learning.

One contribution of this paper is that we have show the advantages of using Partial Max-SAT solvers over weighted Max-SAT solvers when solving problems with hard and soft constraints. On the one hand, we can exploit the learning of modern SAT solvers in the Max-SAT context. As we have seen in the experimental investigation, learning hard clauses produces significant performance improvements on a variety of instances. On the other hand, hard clauses allow to apply a more efficient inference, as well as to compute lower bounds of better quality: (i) the Max-SAT resolution rule is simpler when at least one of the premises is hard; (ii) unit propagation can be enforced on unit hard clauses (while unit propagation on soft clauses is unsound); (iii) a branch of the proof tree can be pruned as soon as a hard clause is violated; (iv) further inconsistencies can be detected in lower bound UP due to the fact that hard clauses used to derive one contradiction can be used again to derive additional contradictions.

Another contribution is that we have defined, to the best of our knowledge, the first learning scheme for soft clauses, and shown that it accelerates the search for an optimal solution on some instances. As we can see in the experiments, it has been particularly useful when solving Partial Max-2-SAT instances. When we look at number of nodes instead of time, we observe that learning soft clauses is superior to learning just hard clauses in a number of instances. We believe that it is worth to design and implement more efficient procedures for learning soft clauses.

It is worth to notice that we have also discussed how Max-SAT resolution can be simplified in the context of Partial Max-SAT, obtaining a complete resolution-style calculus for Partial Max-SAT which is simpler than the calculus for Max-SAT.

As future work, we plan to incorporate into PMS, PMS-hard and PMS-learning additional Max-SAT inference rules like those incorporated into MaxSatz [17] and Toolbar [11], as well as to define new learning schemes for soft clauses. It would be also interesting to evaluate the impact of incorporating into our solvers the non-chronological backtracking and the inference techniques described for Partial Max-SAT in the paper of Heras et al. in this volume [12].

References

1. T. Alsinet, F. Manyà, and J. Planes. Improved exact solver for weighted Max-SAT. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing, SAT-2005, St. Andrews, Scotland*, pages 371–377. Springer LNCS 3569, 2005.
2. J. Argelich and F. Manyà. Solving over-constrained problems with SAT technology. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing, SAT-2005, St. Andrews, Scotland*, pages 1–15. Springer LNCS 3569, 2005.

3. J. Argelich and F. Manyà. Exact Max-SAT solvers for over-constrained problems. *Journal of Heuristics*, 12(4–5):375–392, 2006.
4. J. Argelich and F. Manyà. Learning hard constraints in Max-SAT. In *Proceedings of the Workshop on Constraint Solving and Constraint Logic Porgramming, CSCLP-2006, Lisbon, Portugal*, pages 1–12, 2006.
5. N. Bansal and V. Raman. Upper bounds for MaxSat: Further improved. In *Proc 10th International Symposium on Algorithms and Computation, ISAAC'99, Chennai, India*, pages 247–260. Springer, LNCS 1741, 1999.
6. M. Bonet, J. Levy, and F. Manyà. A complete calculus for Max-SAT. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT-2006, Seattle, USA*, pages 240–251. Springer LNCS 4121, 2006.
7. M. Bonet, J. Levy, and F. Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 2007. doi:10.1016/j.artint.2007.03.001.
8. B. Cha, K. Iwama, Y. Kambayashi, and S. Miyazaki. Local search algorithms for partial MAXSAT. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97, Providence/RI, USA*, pages 263–268. AAAI Press, 1997.
9. Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT-2006, Seattle, USA*, pages 252–265. Springer LNCS 4121, 2006.
10. V. Heinink, M. Seckington, and F. van der Werf. Experiments on Random 2-SoftSAT. Technical report, Delft University of Technology, 2006.
11. F. Heras and J. Larrosa. New inference rules for efficient Max-SAT solving. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-2006, Boston/MA, USA*, pages 68–73, 2006.
12. F. Heras, J. Larrosa, and A. Oliveras. Minimaxsat: A new weighted Max-SAT solver. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing, SAT-2007, Lisbon, Portugal*, 2007.
13. R. G. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.
14. Y. Jiang, H. Kautz, and B. Selman. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *Proceedings of the 1st International Workshop on Artificial Intelligence and Operations Research*, 1995.
15. J. Larrosa and F. Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-2005, Edinburgh, Scotland*, pages 193–198. Morgan Kaufmann, 2005.
16. C. M. Li, F. Manyà, and J. Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP-2005, Sitges, Spain*, pages 403–414. Springer LNCS 3709, 2005.
17. C. M. Li, F. Manyà, and J. Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI-2006, Boston/MA, USA*, pages 86–91, 2006.
18. D. W. Loveland. *Automated Theorem Proving. A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland, 1978.
19. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *39th Design Automation Conference*, 2001.

20. H. Shen and H. Zhang. Study of lower bound functions for max-2-sat. In *Proceedings of AAAI-2004*, pages 185–190, 2004.
21. Z. Xing and W. Zhang. An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(2):47–80, 2005.
22. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *International Conference on Computer Aided Design, ICCAD-2001, San Jose/CA, USA*, pages 279–285, 2001.

MiniMaxSat: A New Weighted Max-SAT Solver

Federico Heras, Javier Larrosa, and Albert Oliveras

Universitat Politecnica de Catalunya,
Jordi Girona 1-3, 08034 Barcelona, Spain

Abstract. In this paper we introduce MINIMAXSAT, a new Max-SAT solver that incorporates the best SAT and Max-SAT techniques. It can handle hard clauses (clauses of mandatory satisfaction as in SAT), soft clauses (clauses whose falsification is penalized by a cost as in Max-SAT) as well as pseudo-boolean objective functions and constraints. Its main features are: learning and backjumping on hard clauses; resolution-based and subtraction-based lower bounding; and lazy propagation with the two-watched literals scheme. Our empirical evaluation on a wide set of optimization benchmarks indicates that its performance is usually close to the best specialized alternative and, in some cases, even better.

1 Introduction

Max-SAT is the optimization version of SAT where the goal is to satisfy the maximum number of clauses. It is considered one of the fundamental combinatorial optimization problems and many important problems can be naturally expressed as Max-SAT. They include academic problems such as *max cut* or *max clique*, as well as real problems in domains like *routing*, *bioinformatics*, *scheduling*, *electronic markets*, *etc...*

There is a long tradition of theoretical work about the structural complexity [1] and approximability [2] of Max-SAT. Most of this work is restricted to the simplest case in which all clauses are equally important (*i.e.*, unweighted Max-SAT) and have a fixed size (mainly binary or ternary clauses). From a practical point of view, a significant progress has been made in the last 3 years [3,4,5,6,7,8]. As a result, there is a handful of new solvers that can deal, for the first time, with medium-sized instances.

The main motivation of our work comes from the study of Max-SAT instances modelling real-world problems. We usually encounter three features:

- The satisfaction of all clauses does not have the same importance, so each clause needs to be associated with a weight that represents the cost of its violation. In the extreme case, which often happens in practice as observed in [9], there are clauses whose satisfaction is mandatory. They are usually modelled by associating a very high weight with them.
- Literals do not appear randomly along the clauses. On the contrary, it is easy to identify patterns, symmetries or other kinds of structures.
- In some problems there are mandatory clauses that reduce dramatically the number of feasible assignments, so the optimization part of the problem only plays a secondary role. However, in some other problems mandatory clauses are trivially satisfiable and the real difficulty lays on the optimization part.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 41–55, 2007.

When we look at current Max-SAT solvers, we find that none of them is robust over these three features. For instance, [7,8] are restricted to formulas in which all clauses are equally important, [3] is restricted to binary clauses, [5] seems to be efficient on very overconstrained problems (*i.e.*, only a small fraction of the clauses can be simultaneously satisfied), while [10] seems to be efficient on slightly overconstrained problems (*i.e.* almost all the clauses can be satisfied). The solver proposed in [11] is the only one that incorporates some learning, so it will presumably perform well on structured problems, but its lower bound computation is relatively weak, so it does not seem to be competitive in pure optimization problems.

In this paper we introduce MINIMAXSAT, a new weighted Max-SAT solver that incorporates the current best SAT and Max-SAT techniques. It is build on top of MiniSAT+ [12], so it borrows its capability to deal with pseudo-boolean problems and all the MiniSAT [13] features processing mandatory clauses such as learning and backjumping. We have extended it allowing it to deal with weighted clauses, while preserving the two-watched literals lazy propagation method. The main original contribution of MINIMAXSAT is that it implements a very efficient lower bounding technique. Specifically, it applies unit propagation in order to detect disjoint inconsistent clauses like in [8] and then it transforms the problem like in [4,14,5] to increment the lower bound. However, while in [4,14,5] only the clauses that accomplish specific patterns are transformed, in MINIMAXSAT there is no need to define such patterns.

The structure of the paper is as follows: Section 2 provides preliminary definitions, Section 3 overviews MINIMAXSAT, Sections 4 and 5 focus on its lower bounding and additional features, respectively. Section 6 reports experimental results and Section 7 presents related work. Finally, Section 8 concludes and points out possible future work.

2 Preliminaries

In the sequel $X = \{x_1, x_2, \ldots, x_n\}$ is the set of boolean variables. A *literal* is either a variable x_i or its negation $\bar{x}_i$. The variable to which literal l refers is noted $var(l)$. Given a literal l, its negation $\bar{l}$ is $\bar{x}_i$ if l is x_i and is x_i if l is $\bar{x}_i$. A *clause* C is a disjunction of literals. In the following, possibly subscripted capital letters A, B, C, D, and E will always represent clauses. The *size* of a clause, noted $|C|$, is the number of literals that it has. The set of variables that appear in C is noted $var(C)$. An *assignment* is a set of literals not containing a variable and its negation. Assignments of maximal size n are called *complete*, otherwise they are called *partial*. An assignment *satisfies* a literal iff it belongs to the assignment, it satisfies a clause iff it satisfies one or more of its literals and it *falsifies* a clause iff it contains the negation of all its literals. In the latter case we say that the clause is *conflicting* as it always happens with the empty clause, noted $\square$.

A *weighted* clause is a pair (C, w), where C is a clause and w is the cost of its falsification, also called its *weight*. Many real problems contain clauses that *must* be satisfied. We call such clauses *mandatory* or *hard* and associate with them a special weight $\top$. Non-mandatory clauses are also called *soft*. A *weighted formula* in *conjunctive normal form* (WCNF) is a multiset of weighted clauses. A *model* is a complete assignment that satisfies all mandatory clauses. The *cost of an assignment* is the sum of weights of the clauses that it falsifies. Given a WCNF formula, *Weighted* Max-SAT is the problem

of finding a model of minimum cost. Note that if a formula contains only mandatory clauses, weighted Max-SAT is equivalent to classical SAT. If all the clauses have weight 1, we have the so-called (unweighted) Max-SAT problem. In the following, we will assume weighted Max-SAT.

We say that formula $\mathcal{F}'$ is a *relaxation* of formula $\mathcal{F}$ (noted $\mathcal{F}' \sqsubseteq \mathcal{F}$) if they are defined over the same set of variables and the cost of any complete assignment in $\mathcal{F}'$ is less than or equal to the cost in $\mathcal{F}$ (non-models are considered to have cost infinity). We say that two formulas $\mathcal{F}'$ and $\mathcal{F}$ are *equivalent* (noted $\mathcal{F}' \equiv \mathcal{F}$) if $\mathcal{F}' \sqsubseteq \mathcal{F}$ and $\mathcal{F} \sqsubseteq \mathcal{F}'$.

If a formula contains clauses (C,u) and (C,v), they can be replaced by $(C,u+v)$ and if it contains a clause $(C,0)$, this may be removed. Both these transformation preserve equivalence. The empty clause may appear in a formula. If its weight is $\top$, it is clear that the formula does not have any model. If its weight is w, the cost of any assignment will include that weight, so w is an obvious lower bound of the formula optimal cost. Weighted empty clauses and their interpretation in terms of lower bounds will become relevant in Section 4.

Mandatory clauses of size 1 (namely, $(l,\top)$) are called *facts*. When a formula contains a fact $(l,\top)$, it can be simplified by removing all clauses containing l and removing $\bar{l}$ from all the clauses where it appears. The application of this rule until quiescence is called *unit propagation* (UP) and it is well recognized as a fundamental propagation technique in all current SAT solvers. Note that most of them use a lazy implementation of UP based on the *two-watched literals scheme* [15].

3 Overview of MiniMaxSat

MiniMaxSat performs a *depth-first branch-and-bound* search on the tree of possible assignments, where internal nodes represent partial assignments and leaf nodes represent complete assignments. Each internal node has two children: the two possible extensions of its associated assignment with respect to one of the unassigned variables. At an arbitrary search point, the algorithm tries to detect a conflict, which means that the current partial assignment cannot be successfully extended. We distinguish two types of conflicts: *hard* conflicts indicate that there is no model extending the current partial assignment (namely, all the mandatory clauses cannot be satisfied), and *soft* conflicts indicate that the current partial assignment cannot be extended to an optimal assignment. Hard conflicts are detected when unit propagation leads to the empty mandatory clause $(\square,\top)$. The detection of soft conflicts requires that the algorithm maintains two values during search:

- The cost of the best model found so far, which is an upper bound *ub* of the optimal solution.
- An underestimation of the best cost that can be achieved extending the current partial assignment into a model, which is a lower bound *lb* of the current subproblem.

A soft conflict is detected when $lb \geq ub$, because it means that the current assignment cannot lead to an optimal model. Note that any soft clause (C,w) with $w \geq ub$ must be satisfied in an optimal assignment. Therefore, in the following we assume that such soft clauses are automatically transformed into hard clauses.

An algorithmic description of MINIMAXSAT is presented in *Algorithm 1*. The algorithm uses a propagation queue Q which contains all facts pending propagation. Once propagated, literals are not removed from Q, but rather marked as such. The algorithm also uses an array $V(l)$ which accumulates the weight of all soft clauses that have become unit over l (namely, clauses $(A \vee l, w)$ such that the current assignment falsifies A).

Before starting the search, a good initial upper bound is obtained with a local search method (line 1) which may yield the identification of some new hard clauses. In our current implementation we use UBCSAT [16] with default parameters. The selected local search algorithm is *IROTS* (*Iterated Robust Tabu Search*). Next, the queue Q is initialized with all the facts in the resulting formula (line 2). The main loop starts in line 3 and each iteration is in charge of propagating all pending facts (line 4) and, if no conflict is detected, attempting the extension of the current partial assignment (line 10). Pending facts in Q are propagated in function `Propagate` (line 4), which may return a hard or soft conflict (see next Section for details). If a hard conflict is encountered (line 5) the conflict is analyzed, a new hard clause is learnt and backjumping is performed. This is done as it is customary in classical SAT solvers such as CHAFF [15]. If a soft conflict is encountered (line 6) chronological backtracking is performed. Note that this does not affect the overall completeness of the procedure, but some subtle implementation details are necessary. If no conflict is found (line 10), a literal is heuristically selected and added to Q for propagation in the next iteration. However, if the current assignment is complete (line 7), the upper bound is updated. Search stops if a zero-cost solution is found because it cannot be further improved (line 8). Else, chronological backtracking is performed (line 9). Note that backjumping leads to termination if a top level hard conflict is found, while chronological backtracking leads to termination if the two values for the first assigned variable have been tried.

Algorithm 2 describes function `Propagate` that performs unit propagation (`UP`) which propagates facts (line 18). It iterates over the non-propagated literals l in Q (line 11). Firstly, the cost of falsifying $\bar{l}$ (which is recorded in $V(\bar{l})$) is added to the lower bound (line 12). Secondly, if a hard clause becomes a fact (line 13), the corresponding literal is added to Q for future propagation (line 14). Finally, if a soft clause becomes unit (q,u) (line 16), its weight u is added to $V(q)$ (line 17). If during this process a hard conflict is detected, the function returns it (line 15). Else, the algorithm attempts to detect a soft conflict with a call to procedure `improveLB` (line 20, see Section 4 for details), and it returns the soft conflict if it is found (line 21). Finally, if no conflict is detected, the function returns *None* (line 22).

Note that `Propagate` only needs to identify when original (soft or hard) clauses have all their literals but one falsified. Thus, we use the *two-watched literals* scheme [15] in both hard and soft clauses. Note that any changes to lb or $V(l)$ must be restored upon backtracking.

4 Lower Bounding in MINIMAXSAT

In the following, we consider an arbitrary search state of MINIMAXSAT before the call to `improveLB`. Such a search state is uniquely characterized by the current assignment. The current assignment determines the *current subformula* which is the

Algorithm 1. MINIMAXSAT basic structure

```
   Function Search() : integer
1      ub := LocalSearch() ;
2      InitQueue(Q) ;
3      Loop
4          Propagate() ;
5          if Hard Conflict then
               Analyze() ;
               if Top Level Hard Conflict then return ub ;
               else
                   LearnClause() ;
                   Backjumping() ;
           else
6              if Soft Conflict then
                   ChronologicalBactracking() ;
                   if End of Search then return ub ;
               else
7                  if all variables assigned then
                       ub := lb ;
8                      if ub = 0 then return ub ;
9                      ChronologicalBactracking() ;
                       if End of Search then return ub ;
10                 else
                       l := SelectLiteral() ;
                       Enqueue(Q,l) ;
```

original formula *conditioned* by the current assignment. The current subformula has the lower bound as the weight of the empty clause $(\square, lb)$. Similarly, value $V(l)$ defines unit clause $(l, V(l))$. Recall that such a unit clause is the aggregation of all the original clauses that have become unit over l due to the current partial assignment.

MINIMAXSAT improves its lower bound in procedure `improveLB` (called in line 20 of *Algorithm 2*). It does so by deriving new soft empty clauses $(\square, w)$ through a weighted resolution process. Such clauses are added to the original $(\square, lb)$ clause producing an increment of the lower bound. *Weighted resolution* (also called *Max-RES*) [4], is a rule that *replaces* two *clashing* clauses $(x \vee A, u)$ and $(\bar{x} \vee B, w)$ by the following set of clauses $\{(A \vee B, m), (x \vee A, u-m), (\bar{x} \vee B, w-m), (x \vee A \vee \bar{B}, m), (\bar{x} \vee \bar{A} \vee B, m)\}$, [1] where $m = \min\{u, w\}$ and hard clauses are treated as if their cost was infinity (*i.e.* $\top - u = \top$). The first clause is called the *resolvent* and the other clauses are called *compensation clauses*. The transformation preserves equivalence as defined in Section 2.

[1] When A is the empty clause, $\overline{A}$ represents a tautology.

Algorithm 2. Functions related with the search algorithm

```
Function UP() : conflict
    while (Q contains non-propagated literals) do
11      l := PickNonPropagatedLiteral(Q); MarkAsPropagated(l) ;
12      lb := lb + V(l̄)) ;
13      foreach Hard clause that has become unit over literal q do
14          Enqueue(Q,q) ;
15          if {q̄} ∈ Q then return Hard Conflict ;
16      foreach Soft clause with weight u that has become unit over literal q do
17          V(q) = V(q) + u ;
    return None ;
Function Propagate() : conflict
18  c := UP() ;
19  if c = Hard Conflict then return c ;
20  improveLB() ;
21  if lb ≥ ub then return Soft Conflict ;
22  return None ;
```

The last two compensation clauses may lose the clausal form, so the following rule [5] may be needed to recover it:

$$CNF(A \vee \overline{l \vee B}, u) = \begin{cases} A \vee \bar{l} & : \quad |B| = 0 \\ \{(A \vee \bar{l} \vee B, u)\} \cup CNF(A \vee \bar{B}, u) & : \quad |B| > 0 \end{cases}$$

Example 1. *If we apply weighted resolution to the following clauses* $\{(x \vee y, 3), (\bar{x} \vee y \vee z, 4)\}$ *we obtain* $\{(y \vee y \vee z, 3), (x \vee y, 3-3), (\bar{x} \vee y \vee z, 4-3), (x \vee y \vee \overline{(y \vee z)}, 3), (\bar{x} \vee \bar{y} \vee y \vee z, 3)\}$. *The first and fourth clauses can be simplified. The second clause can be omitted because it weight is zero. The fifth clause can be omitted because it is a tautology. We apply CNF rule to the fourth clause to obtain two new clauses* $CNF(x \vee y \vee \overline{(y \vee z)}, 3) = \{(x \vee y \vee \bar{y} \vee z), 3), (x \vee y \vee \bar{z}, 3)\}$. *Note that the first new clause is a tautology. Therefore, we obtain the equivalent formula* $\{(y \vee z, 3), (\bar{x} \vee y \vee z, 1), (x \vee y \vee \bar{z}, 3)\}$.

As a first step, `improveLB` performs *unit neighborhood resolution* (UNR) [17,4], which only resolves on pairs of clashing unit clauses. It produces an immediate increment of the lower bound (*i.e.*, the weight of the empty clause) as it is illustrated in the following example,

Example 2. *Consider a search state with two unassigned variables x and y in which the lower bound is* $lb = 3$, $V(x) = V(y) = 1$, $V(\bar{x}) = V(\bar{y}) = 2$ *and a clause* $(x \vee y, 3)$. *This is equivalent to the formula* $\{(\square, 3), (x, 1), (y, 1), (\bar{x}, 2), (\bar{y}, 2), (x \vee y, 3)\}$. *UNR would resolve on clauses* $(x, 1)$ *and* $(\bar{x}, 2)$ *replacing them by* $(\bar{x}, 1)$ *and* $(\square, 1)$ *(all other compensation clauses are removed because their weight is zero or they are tautologies). The two empty clauses can be grouped into* $(\square, 3+1=4)$. *UNR would also resolve on clauses* $(y, 1)$ *and* $(\bar{y}, 2)$ *replacing them by* $(\bar{y}, 1)$ *and* $(\square, 1)$. *The two empty clauses can be grouped into* $(\square, 4+1=5)$. *So, the new equivalent formula is* $\{(\square, 5), (\bar{x}, 1), (\bar{y}, 1), (x \vee y, 3)\}$ *with a higher lower bound of* 5.

As a second step we execute a *simulation of unit propagation* (SUP) in which soft clauses are treated as if they were hard. As seen in the previous section, unit propagation uses a propagation queue Q. In the following, we assume that together with each literal l, the queue Q also contains its *reason*: the clause $(A \vee l, w)$ that cause its unit propagation. If SUP yields a conflict, it means that there is a subset of (soft or hard) clauses that cannot be simultaneously satisfied. Let m be the minimum weight among these clauses. It is easy to see that the extension of the current partial assignment to the unassigned variables will have a cost of at least m. Besides, such a cost can be made explicit by a sequence of resolution steps. A resolution tree is built from the propagation queue Q as follows: let C_0 be the conflicting clause. Traverse Q *from tail to head* until a clashing clause D_0 is found. Then resolution is applied between C_0 and D_0, obtaining resolvent C_1. Next, the traversal of Q continues until a clause D_1 that clashes with C_1 is found, giving resolvent C_2 and we iterate the process until the resolvent we obtain is the empty clause $\square$. Once the resolution tree is computed, weighted resolution can actually be done with the actual soft clauses and, as a result, the empty clause $(\square, m)$ will be derived. Finally, all clauses used in the process will be replaced by $(\square, m)$ and the corresponding compensation clauses, thus obtaining an equivalent formula with a lower bound increment of m. It is important to remark that this transformation preserves equivalence since all clauses are used at most once in the resolution process but we have to undo the transformation upon backtracking. We call this procedure *resolution-based* lower bounding.

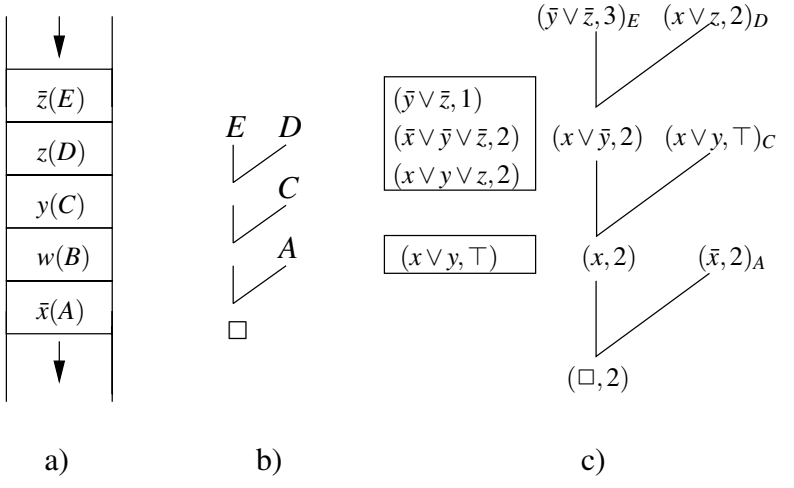
Example 3. *Consider formula* $\mathcal{F} = \{(\bar{x},2)_A, (x \vee w,1)_B, (x \vee y,\top)_C, (x \vee z,2)_D, (\bar{y} \vee \bar{z},3)_E\}$*, where each clause is identified by a subindex for future reference.*

Step 1. Apply SUP. *Initially, the unit clause A is enqueued producing* $Q = [\bar{x}(A)]$ *(within parenthesis, we indicate the reason of a literal). Then* $\bar{x}$ *is propagated. The resulting formula is* $\{(w,1)_B, (y,\top)_C, (z,2)_D, (\bar{y} \vee \bar{z},3)_E\}$ *and Q becomes* $[\bar{x}(A), w(B), y(C), z(D)]$*. Literal w is propagated. The resulting formula is* $\{(y,\top)_C, (z,2)_D, (\bar{y} \vee \bar{z},3)_E\}$ *and no new unit clauses are generated. Literal y is propagated. The resulting formula is* $\{(z,2)_D, (\bar{z},3)_E\}$ *and a new unit clause is enqueued producing* $Q = [\bar{x}(A), w(B), y(C), z(D), \bar{z}(E)]$*. Since z and* $\bar{z}$ *are inside Q, a conflict is detected and SUP stops. Note that E is the conflicting clause. Figure 1.a shows the state of Q after the propagation.*

Step 2. Build the resolution tree. *Starting from the tail of Q the first clause clashing with the conflicting clause E is D. Resolution between E and D generates the resolvent* $x \vee \bar{y}$*. The first clause clashing with it is C, producing resolvent x. The next clause clashing with it is A and resolution generates* $\square$*. Figure 1.b shows the resulting resolution tree. The minimum weight among the involved clauses is* 2*.*

Step 3. Transform the problem. *We apply weighted resolution as indicated by the tree computed in Step 2. Figure 1.c graphically shows the result of the process. Leaf clauses are the original clauses involved in the resolution. Each internal node indicates a resolution step. The resolvents appear in the junction of the edges. Beside each resolvent, inside a box, there are the compensation clauses that must be added to the formula to preserve equivalence. Since clauses that are used in resolution must be removed, the resulting formula* $\mathcal{F}'$ *consists of the root of the tree (*$(\square, 2)$*) and all compensation clauses. That is, the resulting formula is* $\mathcal{F}' = \{(x \vee w,1), (x \vee y,\top), (\bar{y} \vee \bar{z},1), (\square,2), (x \vee y \vee z,2), (\bar{x} \vee \bar{y} \vee \bar{z},2)\}$*. Note that* $\mathcal{F} \equiv \mathcal{F}'$*.*

$$\mathcal{F} = \{(\bar{x},2)_A, (x \vee w,1)_B, (x \vee y,\top)_C, (x \vee z,2)_D, (\bar{y} \vee \bar{z},3)_E\}$$



$$\mathcal{F}' = \{(x \vee w,1), (x \vee y,\top), (\bar{y} \vee \bar{z},1), (\square,2), (x \vee y \vee z,2), (\bar{x} \vee \bar{y} \vee \bar{z},2)\}$$
$$\mathcal{F}'' = \{(x \vee w,1), (x \vee y,\top), (\bar{y} \vee \bar{z},1), (\square,2)\}$$

Fig. 1. Graphical representation of MINIMAXSAT lower bounding. On the top, the original formula $\mathcal{F}$. On the left, the propagation Q after step 1. In the middle, the structure of the resolution tree computed in step 2. On the right, the effect of actually executing the resolution (step 3). The resulting formula $\mathcal{F}'$ appears bellow. If subtraction-based lower bounding is performed, step 3 is replaced by a subtraction of weights, producing formula $\mathcal{F}''$.

An alternative to problem transformation through resolution is to identify the lower bound increment m and then subtract it from all the clauses that would have participated in the resolution tree. This procedure is reminiscent of the lower bound computed in [7] and we call it *subtraction-based* lower bounding.

Example 4. *Consider formula $\mathcal{F}$ from the previous example. Steps 1 and 2 are identical. However, subtraction-based lower bounding would replace Step 3 by Step 3' that subtracts weight 2 from the clauses that appear in the resolution tree and then adds $(\square,2)$ to the formula. The result is $\mathcal{F}'' = \{(x \vee w,1), (x \vee y,\top), (\bar{y} \vee \bar{z},1), (\square,2)\}$. Note that $\mathcal{F}'' \sqsubseteq \mathcal{F}$, so its lower bound is also a lower bound of $\mathcal{F}$, but they are not equivalent. Hence, $\mathcal{F}''$ cannot be used in the subsequent search and if no soft conflict is immediately detected, this transformation has to be undone before continuing the search.*

After the increment of the lower bound with either technique, procedure SUP can be executed again, which may yield new lower bound increments. The process is repeated until SUP does not detect any conflict.

When comparing the two previous approaches, we find that resolution-based lower bounding has a larger overhead, because resolution steps need to be actually computed

and their consequences must be added to the current formula and removed upon backtracking. However, the effort invested in the transformation is usually amortized because the increment obtained in the lower bound *becomes part of the current formula*, so it does not have to be *discovered* again and again by all the descendent nodes of the search as it would happen with the subtraction-based approach. In our experiments, we found that no scheme was systematically better than the other. We also found that resolution-based lower bounding seems to be more effective if resolution is only applied to low arity clauses. As a consequence, after the identification of the resolution tree, MINIMAXSAT only applies resolution-based lower bounding if the largest resolvent in the resolution tree has arity less than 4. Observe that compensation clauses will contain at most 4 literals. Otherwise, it applies subtraction-based lower bounding.

5 Additional Features of MINIMAXSAT

5.1 Probing

Probing is a well-known SAT technique that allows the formulation of hypothetical scenarios [18]. The idea is to temporarily assume that l is a fact and then execute unit propagation. If UP yields a conflict, we know that $\bar{l}$ is indeed a fact. The process is iterated over all the literals until quiescence. Exhaustive experiments in the SAT context indicate that it is too expensive to probe during search, so it is normally done as a preprocess in order to reduce the initial number of branching points.

We can easily extend this idea to Max-SAT. In that context, besides the *discovery* of facts, it may be used to make explicit weighted unit clauses. As in SAT, the idea is to temporarily assume that l is a fact and then *simulate* unit propagation (*i.e.*, execute SUP()). Then, we build the resolution tree T from the propagation queue Q. If all the clauses in T are hard, we know that $\bar{l}$ is indeed a fact. Else, we can reproduce T applying Max-RES with the actual clauses and derive a unit clause $(\bar{l},m)$ where m is the minimum weight among the clauses in T. Having unit soft clauses upfront makes the future executions of `improveLB` much more effective in the subsequent search. Besides, if we derive both (l,u) and $(\bar{l},w)$, we can generate via unit neighborhood resolution (see Example 2) an initial non-trivial lower bound of $min\{u,w\}$. We tested probing during search and as a preprocessing in several benchmarks. We observed empirically that probing as a preprocessing was the best option as it is in SAT.

Example 5. *Consider formula* $\mathcal{F} = \{(x \vee y,1)_A,(x \vee z,1)_B,(\bar{y} \vee \bar{z},1)_C\}$. *If we assume* $\bar{x}$ *by adding it to Q and then execute SUP a conflict is reached. We obtain* $Q = [\bar{x}(\varnothing), y(A), z(B), \bar{z}(C)]$ *and we detect that C is a conflicting clause. The clauses involved in the refutation are C, B, and A. Resolving clauses C and B results in* $\{(x \vee y,1)_A,(x \vee \bar{y},1),(x \vee y \vee z,1),(\bar{x} \vee \bar{y} \vee \bar{z},1)\}$. *The resolution of the previous resolvent and A produces the (equivalent) formula* $\mathcal{F}' = \{(x,1),(x \vee y \vee z,1),(\bar{x} \vee \bar{y} \vee \bar{z},1)\}$.

5.2 Branching Heuristic

For problems where all literals appear in hard clauses in only one polarity, a weighted version of the *Two-sided Jeroslow Wang* heuristic [14] is computed in the root node

and used in the subsequent search (the importance of each clause is multiplied by its weight). For problems where literals appear in hard clauses with both polarities it applies the native VSIDS-like heuristic [15] of MiniSat. In both cases, if some literal l accomplishes $V(l) + lb \geq ub$ at some node of the search tree, then $\bar{l}$ is the selected literal to assign and l is never assigned.

5.3 Pseudo-boolean Optimization

MINIMAXSAT can solve *pseudo-boolean optimization problem* [19,12] of the form:
(1) minimize $\sum_{j=1}^{n} c_j \cdot x_j$
(2) subject to $\sum_{j=1}^{n} a_{ij} l_j \geq b_i, \quad i = 1 \ldots m$

where $x_j \in \{0,1\}$, l_j is either x_j or $1 - x_j$, and c_j, a_{ij} and b_i are non-negative integers. (1) is the *objective function* and (2) is the set of *pseudo-boolean constraints*. MINIMAXSAT uses MINISAT+ to transform pseudo-boolean constraints into hard clauses. That is, it determines heuristically the most appropriate encoding to hard clauses through *adders*, *sorters* or *BDDs*. Regarding the objective function, for each pair $c_j \cdot x_j$, a new soft unit clause $(\bar{x}_j, c_j)$ is added.

6 Experimental Results

We compare MINIMAXSAT with several optimizers from different communities:

- MAXSATZ [8,20]. Specialized unweighted Max-SAT solver. It applies a powerful subtraction lower bounding [8] plus limited transformation rules [20].
- MAX-DPLL [14,5] is a Weighted Max-SAT solver that performs a restricted form of resolution lower-bounding. MAX-DPLL is part of the TOOLBAR package.
- TOOLBAR [17,21,22,23]. It is a state-of-the-art Weighted CSP solver.
- PUEBLO 1.5 [19] is a pure pseudo-boolean solver.
- MINISAT+ [12] is a pseudo-boolean solver that translates pseudo-boolean problems into SAT and solves them with MiniSAT.

When reporting results, we will omit a solver if it cannot deal with the corresponding instances or it performs extremely bad. Results are presented in plots and tables. The first column of each table contains the name of the set of problems and the second shows the number of instances. The rest of columns report the performance of the solvers. Each cell contains the average CPU time that the solver required to solve all instances. If not all the instances were solved within the time limit (600 seconds), a number inside brackets indicates the number of solved instances and the average CPU time only takes into account solved instances. Note that in the plots the order of the legend goes in accordance with the performance of the solvers. All experiments were conducted on a 3.2 Ghz Pentium 4 computer with Linux.

The following benchmarks were considered:

- Random Max-2-SAT instances with 100 variables and clauses ranging from 200 to 900. Max-3-SAT instances with 80 variables and clauses ranging from 300 to 700. See [14].

- Random Max-CUT instances [14] with 60 nodes and the number of edges ranging from 300 to 500.
- Random and structured Max-Clique instances [5]. The random instances have 150 nodes and the edge density is ranged from 0 to 100 per cent. The structured instances correspond to the 66 instances of Dimacs Challenge.
- Combinatorial Auctions [5]. The instances were generated with CATS [24]. Three distributions were considered: *paths*, *scheduling* and *regions*. The number of goods is fixed to 60 and the number of bids is varied differently for each distribution.
- Max-One instances [5]. We have selected some SAT instances for which we have solved the Max-One problem. We considered structured instances coming from the 2002 SAT Competition [11] and random 3-SAT instances with 120 variables and ranging the number of clauses from 150 to 550.
- WCSP instances. Structured Planing instances [25] containing both boolean and non-boolean variables and hard and soft constraints. Random binary Max-CSP instances have three or four values per variable and only soft constraints. Depending on the number of constraints and the number of forbidden tuples, 4 distributions were generated: Dense Loose (DL), Dense Tight (DT), Sparse Loose (SL) and Sparse Tight (ST) [21]. WCSP instances were translated to Weighted Max-SAT using the direct encoding [26].
- Small integer optimization pseudo-boolean instances coming from the 2006 Pseudo-Boolean Evaluation. We considered some industrial instances corresponding to *logic synthesis* , and some handmade instances including *Misc* (*garden*), *min prime* and *MPS* (*miplib*).

Figure 2 contains plots with the results on different benchmarks. Plots *a* and *b* reports results on random unweighted Max-SAT instances. PUEBLO and MINISAT+ are orders of magnitude slower, so they are not included in the graphics. On Max-2-SAT (plot *a*), MINIMAXSAT lays between MAX-DPLL and MAXSATZ, which is the best option. On Max-3-SAT (plot *b*) MINIMAXSAT clearly outperforms MAX-DPLL and is very close to MAXSATZ, which is again the best. In both Max-2-SAT and Max-3-SAT MAXSATZ is no more than 3 times faster than MINIMAXSAT. Plot *c* reports results on Max-CUT instances. In these problems, MINIMAXSAT performs slightly better than MAXSATZ, which is the second alternative.

Plot *e* reports the results on Random Max-Clique instances. MINIMAXSAT is the best solver, up to an order of magnitude faster than MAX-DPLL, the second option. PUEBLO and MINISAT+ perform poorly again. Regarding the structured Dimacs instances, MINIMAXSAT is again the best option. It solves 34 instances within the time limit, while TOOLBAR,MINISAT+ and PUEBLO solve 29, 19 and 14 respectively.

Plots *f*, *g* and *h* present the results on Combinatorial Auctions following different distributions. On the paths distribution, MINIMAXSAT is the best solver, twice faster than MAX-DPLL, which ranks second. On the regions distribution, MAX-DPLL is the best solver while MINIMAXSAT is the second best solver requiring double time. On the paths and regions distributions, PUEBLO and MINISAT+ perform very poorly. On the scheduling distribution, MINISAT+ is the best solver while MAX-DPLL and MINIMAXSAT are about one order of magnitude slower.

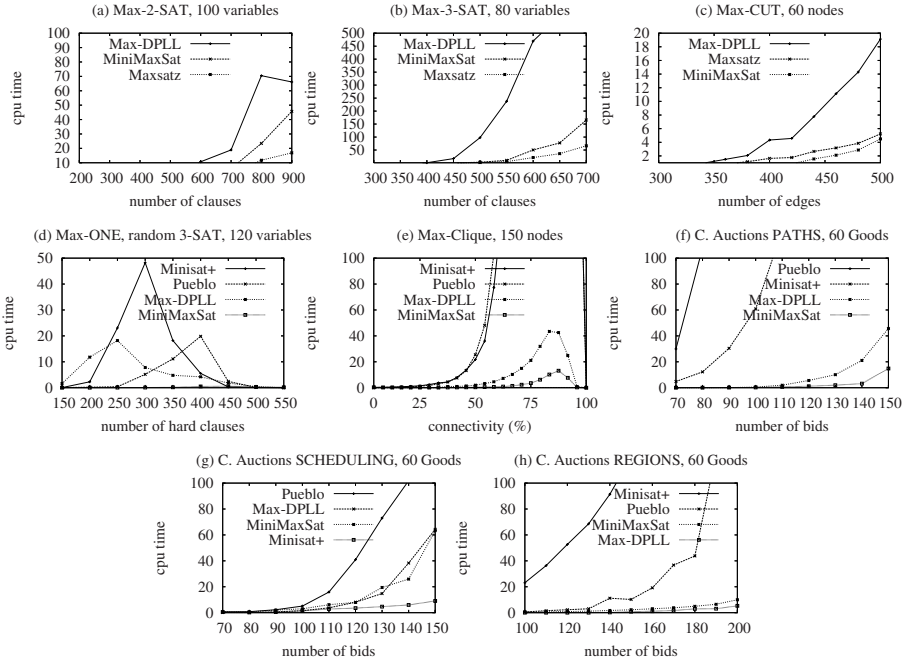


Fig. 2. Plots of different benchmarks. Note that the order in the legend goes in accordance with the performance of the solvers.

On random Max-One (plot *d*) MINIMAXSAT is the best solver by far. Almost all instances are solved instantly while PUEBLO and MAX-DPLL require up to 20 seconds in the most difficult instances. MINISAT+ performs very poorly. The results on structured Max-One instances are reported in Figure 3. MINISAT+ seems to be the fastest in general. MINIMAXSAT is close in performance to PUEBLO. Note, however, that in the *dp* instances, MINIMAXSAT is the system solving more instances.

On structured Planning WCSP instances (Fig. 4) PUEBLO is the best solver. MINIMAXSAT is the second best solver, TOOLBAR is the third and the last one is MINISAT+. This is not surprising since TOOLBAR does not perform learning over the hard constraints. However, on pure optimization Max-CSP problems (Fig. 4) TOOLBAR solves all the instances instantly while PUEBLO performs very poorly. MINIMAXSAT is clearly the second best solver on DL instances, while MINISAT+ is the second best option on DT and ST tight instances.

Results regarding pseudo-boolean instances can be found in Figure 5. Note that this is the first time that a Max-SAT solver is tested on pseudo-boolean instances. Results indicate that no solver consistently outperforms the other and that MINIMAXSAT is fairly competitive with PUEBLO and MINISAT+.

We can conclude that MINIMAXSAT is the most robust Weighted Max-SAT solver. It is very competitive for pure optimization problems and for problems with lots of hard clauses and, sometimes, it is the best option.

Problem	n. inst.	MINIMAXSAT	Pueblo	Minisat+
3col80	10	0.25	0.15	0.05
3col100	10	2.90	2.55	0.26
3col120	10	28.77	21.23	1.50
3col140	10	56.57	122.59	3.86
cnt	3	9.30	0.25	0.25
dp	6	11.75(5)	1.82(3)	2.40(4)
ezfact32	10	1.49	0.69	0.65

Fig. 3. Structured Max-one instances

Problem	n. inst.	Toolbar	MINIMAXSAT	Pueblo	Minisat+
Planning	71	8.22	2.19	0.28	13.64
DL	20	0.14	2.20	302.85(8)	27.17
DT	20	0.00	7.48	0(0)	5.33
SL	20	0.01	33.08	83.30(18)	1.30
ST	20	0.00	18.04	0(0)	4.29

Fig. 4. Results for WCSP instances

Problem	n. inst.	MINIMAXSAT	Pueblo	Minisat+
Garden	7	2.87(5)	13.60(5)	0.28(5)
Logic synthesis	17	26.33(2)	57.60(5)	4.21(2)
Min prime	156	20.94(111)	13.20(106)	7.58(112)
Miplib	17	34.50(5)	51.84(9)	21.48(9)

Fig. 5. Results for pseudo-boolean instances

7 Related Work

Some previous work has been done about incorporating SAT-techniques inside a Max-SAT solver. In [10] a lazy data structure to detect when clauses become unit is presented but it requires a static branching heuristic, so it is not as general as our extension of the two-watched literals. As far as we know, the rest of Max-SAT solvers are based on *adjacency lists* that are inefficient for unit propagation [27]. In [11] a Max-SAT branch and bound is powered with learning over hard constraints, but it is used in combination of simple lower bounding techniques. To the best of our knowledge, no Max-SAT solver incorporates backjumping. Note that MINIMAXSAT restricts backjumping to the ocurrence of hard conflicts. Related frameworks that backjump after soft conflicts include [28] for *WCSP*, [29] for pseudo-boolean optimization and [30] for *SMT*.

Most Max-SAT solvers use what we call subtraction-based lower bounding. In most cases, they search for special patterns of mutually inconsistent subsets of clauses [3,6,10]. For efficiency reasons, these patterns are always restricted to small sets of small arity clauses (2 or 3 clauses or arity less than 3). MINIMAXSAT uses a natural weighted extension of the approach proposed in [7]. It was the first one able to detect inconsistencies in arbitrarily large sets of arbitrarily large clauses.

The idea of what we call resolution-based lower bounding was inspired from the WCSP domain [17,21,22,23] and it was first proposed in the Max-SAT context in [4] and further developed in [20,14,5]. In these works, only special patterns of fixed-size resolution trees were executed. The use of simulated unit propagation allows MINIMAXSAT to identify arbitrarily large resolution trees.

Our probing method to derive weighted unit clauses is related to the $2 - RES$ and cycle rule of [14,5] and to failed literals in [8]. Again, the use of simulated unit propagation allows MINIMAXSAT to identify arbitrarily large resolution trees.

8 Conclusions and Future Work

MINIMAXSAT is an efficient and very robust Max-SAT solver that can deal with hard and soft clauses as well as pseudo-boolean functions. It incorporates the best techniques for each type of problems, so its performance is similar to the best specialized solver. Besides the development of MINIMAXSAT combining, for the first time, known techniques from different fields, the main original contribution of this paper is a novel lower bounding technique based on resolution. MINIMAXSAT lower bounding subsumes in a very clean an elegant way most of the approaches that have been proposed in the last years. Future work concerns the development of VSIDS-like heuristics for soft clauses, backjumping techniques for soft conflicts and the study of domain-specific branching heuristics.

References

1. Papadimitriou, C.: Computational Complexity. Addison-Wesley, USA (1994)
2. Karloff, H.J., Zwick, U.: A 7/8-Approximation Algorithm for MAX 3SAT. In: FOCS. (1997)
3. Shen, H., Zhang, H.: Study of lower bounds for Max-2-SAT. In: AAAI. (2004)
4. Larrosa, J., Heras, F.: Resolution in Max-SAT and its relation to local consistency for weighted CSPs. In: IJCAI. (2005)
5. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-sat solving. In: Available at the Computing Research Repository (http://arxiv.org/PS_cache/cs/ps/0611/0611025.ps.gz). (2006)
6. Xing, Z., Zhang, W.: MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability. Artificial Intelligence **164** (2005) 47–80
7. Chu Min Li, F.M., Planes, J.: Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers. In: Proc. of the 11^{th} CP, Sitges, Spain (2005)
8. Li, C.M., Manyà, F., Planes, J.: Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In: AAAI. (2006)
9. Cha, B., Iwama, K., Kambayashi, Y., Miyazaki, S.: Local search algorithms for partial MAXSAT. In: AAAI/IAAI. (1997) 263–268
10. Alsinet, T., Manya, F., Planes, J.: Improved exact solver for weighted max-sat. In SAT'05.
11. Argelich, J., Manyà, F.: Learning hard constraints in max-sat. In: CSCLP-2006. (2006) 1–12
12. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into sat. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006) 1–26
13. Eén, N., Sörensson, N.: An extensible sat-solver. In: Proceedings of SAT03. (2003) 502–518
14. Heras, F., Larrosa, J.: New inference rules for efficient max-sat solving. In: AAAI. (2006)
15. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: DAC. (2001) 530–535
16. Tompkins, D.A.D., Hoos, H.H.: Ubcsat: An implementation and experimentation environment for sls algorithms for sat & max-sat. In: SAT. (2004)
17. Larrosa, J.: Node and arc consistency in weighted CSP. In: AAAI. (2002) 48–53
18. Lynce, I., Silva, J.P.M.: Probing-based preprocessing techniques for propositional satisfiability. In: ICTAI. (2003) 105–
19. Sheini, H.M., Sakallah, K.A.: Pueblo: A hybrid pseudo-boolean sat solver. Journal on Satisfiability, Boolean Modeling and Computation **2** (2006) 165–189
20. Li, C.M., Manyà, F., Planes, J.: New inference rules for max-sat. In: Submitted. (2006)
21. Larrosa, J., Schiex, T.: In the quest of the best form of local consistency for weighted CSP. In: Proc. of the 18^{th} IJCAI, Acapulco, Mexico (2003)

22. de Givry, S., Larrosa, J., Meseguer, P., Schiex, T.: Solving max-SAT as weighted CSP. In: Proc. of the 9^{th} CP, Kinsale, Ireland, LNCS 2833. Springer Verlag (2003) 363–376
23. de Givry, S., Heras, F., Larrosa, J., Zytnicki, M.: Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In: Proc. of the 19^{th} IJCAI, Edinburgh, U.K. (2005)
24. K. Leyton-Brown, M.P., Shoham, Y.: Towards a universal test suite for combinatorial auction algorithms. ACM E-Commerce (2000) 66–76
25. Cooper, M., Cussat-Blanc, S., de Roquemaurel, M., Régnier, P.: Soft arc consistency applied to optimal planning. In: CP. (2006) 680–684
26. Walsh, T.: SAT v CSP. In: CP. (2000) 441–456
27. Lynce, I., Silva, J.P.M.: Efficient data structures for backtrack search sat solvers. Ann. Math. Artif. Intell. **43** (2005) 137–152
28. Zivan, R., Meisels, A.: Conflict directed backjumping for maxcsps. In: IJCAI. (2007)
29. Manquinho, V.M., Silva, J.P.M.: Satisfiability-based algorithms for boolean optimization. Ann. Math. Artif. Intell. **40** (2004) 353–372
30. Nieuwenhuis, R., Oliveras, A.: On SAT Modulo Theories and Optimization Problems. In: SAT. (2006) 156–169

Solving Multi-objective Pseudo-Boolean Problems*

Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich

Hardware-Software-Co-Design
Department of Computer Science 12
University of Erlangen-Nuremberg, Germany
{martin.lukasiewycz,glass,haubelt,teich}@cs.fau.de

Abstract. Integer Linear Programs are widely used in areas such as routing problems, scheduling analysis and optimization, logic synthesis, and partitioning problems. As many of these problems have a Boolean nature, i.e., the variables are restricted to 0 and 1, so called *Pseudo-Boolean solvers* have been proposed. They are mostly based on SAT solvers which took continuous improvements over the past years. However, Pseudo-Boolean solvers are only able to optimize a single linear function while fulfilling several constraints. Unfortunately many real-world optimization problems have multiple objective functions which are often conflicting and have to be optimized simultaneously, resulting in general in a set of optimal solutions. As a consequence, a single-objective Pseudo-Boolean solver will not be able to find this set of optimal solutions. As a remedy, we propose three different algorithms for solving multi-objective Pseudo-Boolean problems. Our experimental results will show the applicability of these algorithms on the basis of several test cases.

1 Introduction

Solving *0-1 Integer Linear Programs* (0-1 ILP) came to the field of vision over the past years. This problem class is a special case of *Integer Linear Programs* (ILP) and is also termed as *Pseudo-Boolean* (PB) [1]. In particular a Pseudo-Boolean problem is an optimization problem with a linear objective function and a set of linear constraints in which the coefficients are integers and the variables are restricted to 0 and 1. Despite the restriction of the variables to Boolean values the expressiveness is equal to ILPs which can be formulated as Pseudo-Boolean problems by using a binary encoding.

The Boolean nature of Pseudo-Boolean problems is connecting these strongly to the *Satisfiability problem* (SAT) in conjunctive normal form [2]. The Satisfiability problem can easily be converted to a Pseudo-Boolean problem with an empty objective function in which for each clause a greater-zero constraint is added. The 0-1 Integer Linear Programming is, in fact, one of KARP'S 21 NP-complete problems [3]. On the other hand, converting efficiently PB constraints into clauses is a non-trivial problem that can result in an exponential number of clauses.

There are several PB solvers that are borrowing techniques from state-of-the-art SAT solvers which became essential in the field of *Electronic Design Automation* [4]. These specialized PB solvers are based on the DPLL backtracking algorithm [5] and benefit

* Supported in part by the German Science Foundation (DFG), SFB 694.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 56–69, 2007.

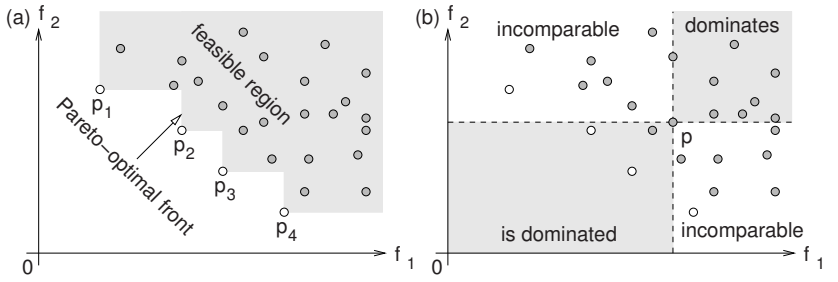


Fig. 1. Objective space showing (a) a Pareto-optimal front of solutions and (b) a solution p and the areas with dominating, non-dominating and incomparable solutions

from the improvements on the field of SAT-solving of the recent years like the non-chronological backtracking [6], watched literals [7], or an efficient conflict learning scheme [8]. As a matter of fact it is validated that specialized PB solvers are superior to generic ILPs, mostly if the underlying problem has a Boolean nature [9].

PB solvers have their applications among many real-world applications like routing problems, scheduling analysis and optimization, logic and system level synthesis, and partitioning problems. Some of these applications like the problem of system level synthesis [10] can contain more than one objective function, e.g., if the system is optimized by its power consumption, area usage, and the monetary costs. In the case of multi-objective optimization the goal is not to find optimal solutions corresponding to each objective function, but to find the set of optimal solutions the so called Pareto-optimal solutions. A solution is called Pareto-optimal if there exists no other solution that is better or equal in all objectives and at least better in one objective, i.e., no other solution *dominates* the Pareto-optimal one. As the search space in Pseudo-Boolean problems is finite the number of Pareto-optimal solutions is also finite. Figure 1(a) illustrates the Pareto-optimal solutions of a problem with two objective functions. A PB solver optimizes at most one objective function and will not find these Pareto-optimal solutions as preference-based approaches do not find the trade-off solutions. In the case of system level synthesis a designer is interested in the full set of Pareto-optimal solutions containing the trade-off solutions to make an appropriate choice for one implementation.

This paper is dedicated to the multi-objective Pseudo-Boolean problem in which we propose three different algorithms for solving multi-objective Pseudo-Boolean problems and compare them on the basis of several test cases. The first algorithm is an iterative search with a common PB solver by restricting the search space by upper bounds. The second algorithm extends a DPLL backtracking algorithm such that it sifts through the valid search space and at the same time prunes evidently not optimal solutions. The third algorithm is using a translation into the Satisfiability problem such that a common SAT solver finds one solution that fulfills the constraints. To ensure a convergence to the Pareto-optimal solutions, the found and dominated solutions are excluded from the ongoing search by appending additional clauses.

The rest of the paper is organized as follows: Section 2 gives a short introduction to the functionality of modern PB solvers, and Section 3 will formally state the problem this paper is dedicated to. In Section 4 the three algorithms for solving multi-objective

Pseudo-Boolean problem are presented. The comparison of the algorithms on the basis of several experimental results is made in Section 5, before we conclude the paper in Section 6.

2 Specialized PB Solvers

Mathematically a Pseudo-Boolean problem is defined as[1]

$$min\{c^T x \mid Ax \leq b\}$$

with $c \in \mathbb{Z}^n$, $A \in \mathbb{Z}^{m,n}$, $b \in \mathbb{Z}^m$, and $x \in \{0,1\}^n$. The objective function is given as the linear function $c^T x$, whereas the constraints that are linear equalities and inequalities are summarized in $Ax \leq b$. The goal is to find one optimal solution x for which the objective function is minimal.

Specialized PB solvers are based on a backtracking search algorithm similar to modern SAT solvers. The algorithm starts by searching for a solution that fulfills all constraints. If there exists a solution x the objective function is calculated and a '$<$' constraints is added with left hand side the objective function and right hand side the calculated objective value. This procedure is carried out iteratively and ensures the convergence to the optimal value. If the constraints are not satisfiable the last found solution is the optimal solution.

These specialized PB solvers are divided into two categories: First are enhanced SAT solvers that beside clauses in the conjunction normal form also support natively PB constraints, e.g., PBS [9], PUEBLO [11], or GALENA [12]. The second category are PB solvers which translate the PB constraints into clauses such that a common SAT solver is used to find a solution. By introducing additional variables an exponential number of resulting clauses is prevented. The proceeding as described in [13] is two-staged and is implemented in the PB solver MINISAT+ [13,14]. Each PB constraint is first converted into a hardware circuit by using BDDs, Adders, or Sorters. The resulting hardware circuits are then converted linearly into a set of clauses by using the TSEITIN-transformation [15] that introduces additional variables.

3 Problem Formulation

Extending a 0-1 ILP for multiple objective functions results in a new problem class. Mathematically we define a multi-objective Pseudo-Boolean problem as

$$min\{C^T x \mid Ax \leq b\}$$

with $C \in \mathbb{Z}^{n,z}$, $A \in \mathbb{Z}^{m,n}$, $b \in \mathbb{Z}^m$, and $x \in \{0,1\}^n$. We are considering an optimization problem with z objective functions which, without loss of generality, all have to be minimized. The objective vectors are calculated by $f(x) = C^T x$ with $C = (c_1, ..., c_z)$ where a single objective function is $f_i(x) = c_i^T x$ with $i \in \{1, ..., z\}$.

The optimization for a multi-objective problem is not a search for a single optimal objective value but instead for the set of Pareto-optimal solutions $X_p \subseteq X_f$ or the Pareto-optimal front $Y_p = \{f(x) | x \in X_p\}$, respectively. The valid search space X_f is containing

[1] Maximization problems can be converted to a minimization by negating the objective function.

all solutions that are fulfilling the constraints $Ax \leq b$, we are also speaking of *feasible* solutions. A solution $x_p \in X_p$ is said to be Pareto-optimal if its objective vector $f(x_p)$ is not dominated by any other objective vector $f(x)$ with $x \in X_f$, cf. Definition 1.

Relating to Definition 1 the terms for Pareto dominance are applied to the objective vector or the solution vectors, respectively. The solution vectors are termed by Definition 1 with respect to their calculated objective vectors, e.g., x dominates $\tilde{x}$ if and only if $f(x) \succ f(\tilde{x})$.

Definition 1 (Pareto dominance (cf. [16])). *For any two objective vectors a and b,*

$$\begin{aligned} a \succ\succ b \;\; &(a \textit{ strictly dominates } b) \textit{ if } \forall i : a_i < b_i \\ a \succ b \;\; &(a \textit{ dominates } b) \textit{ if } \forall i : a_i \leq b_i \wedge \exists j : a_j < b_j \\ a \succeq b \;\; &(a \textit{ weakly dominates } b) \textit{ if } \forall i : a_i \leq b_i \\ a \parallel b \;\; &(a \textit{ is incomparable to } b) \textit{ if } \exists i, j : a_i > b_i \wedge a_j < b_j. \end{aligned}$$

In general, a common iteratively working PB solver as described in Section 2 will not be able to find the set of Pareto-optimal solutions of the multi-objective Pseudo-Boolean problem. By adding a PB constraint each time a solution is found these PB solvers are restricting the search space such that weakly dominated solutions are not found in the ongoing search. Using this approach for a multi-objective problem it would be necessary to add a PB constraint for each objective. These constraints would have to be joined by a logical OR as an improvement in only one dimension is mandatory to find the next not dominated solution. It is obvious that a common PB solver can not be simply adapted to solve multi-objective Pseudo-Boolean problems as all constraints are joined by a logical AND unless additional variables are added for each iteration.

4 Algorithms

4.1 Algorithm 1

The first algorithm, given in Algorithm 1, enables the usage of a common PB solver to find all Pareto-optimal solutions iteratively. In order to find these solutions the upper bounds for the objective functions are set adequately as constraints for the PB solver.

The algorithm fills the *archive* A with the non-dominated solutions. These are solutions that are not dominated by any other solution that was found during the ongoing search. The property of non-dominance proves that there is no solution inside the archive that is better or equal in all objectives compared to another solution in the archive, cf. Definition 1 on *weak domination*. The set D contains the *domains* where a domain is a vector of the upper bounds for the objective functions. The algorithm starts with an empty archive (line 1) and the set of domains containing an initial vector of the length z, where the initial values are set to ∞ (line2) corresponding the whole search space.

While the set of domains is not empty (line 3) one domain is chosen randomly (line 4) and a solution that both fulfills the constraints and is located inside the selected domain is searched with a common PB solver (line 5). At the same time the objective function is empty. If the PB solver does not find a solution, the current domain is removed from the set of domains since there is no feasible solution inside (line 6,7). In case that a solution

Algorithm 1. Algorithm for multi-objective optimization of Pseudo-Boolean problems based on the iterative usage of a common PB solver

```
1  A = {}
2  D = {(max_1, ..., max_z)}
3  while |D| > 0 do
4      choose h ∈ D
5      min{0 | Ax ≤ b ∧ C^T x ≤ h}
6      if UNSATISFIABLE then
7          D = D\h
8      else
9          y = C^T x
10         A = x ∪ {a | a ∈ A ∧ ¬(y ≻ C^T a)}
11         foreach e ∈ D ∧ y ⪰ e do
12             D = D\e
13             for i ∈ {1, ..., z} do
14                 D = D ∪ (e_1, ..., y_i − 1, ..., e_z)
15             end
16         end
17         foreach e, ẽ ∈ D ∧ e ≱ ẽ ∧ e ⪰ ẽ do
18             D = D\e
19         end
20     end
21 end
```

is found in the domain, the objective vector is calculated (line 9) and the archive is updated such that it just contains non-dominated solutions (line 10).

Each domain that contains the found solution needs to be split (line 11). This is done by removing that domain and adding new domains whereas an improvement in at least one dimension has to be achieved (line 12-15). Concluding, the set D is cleaned up in which sub-domains of other domains are removed (line 17-19).

The advantage of this methodology is the independence of the used PB solver. Any common PB solver can be extended to a multi-objective PB solver by this algorithm. Moreover, this straightforward method is not restricted to Pseudo-Boolean problems, by using an ILP solver it is possible to solve also multi-objective Integer Linear Programs.

4.2 Algorithm 2

The second method is an extension of the DPLL backtracking algorithm. More precisely, it is a modification of the DPLL algorithm as it is used in specialized PB solvers. Thereby, it is not important which category of specialized PB solver is used, the ones that natively support PB constraint or the other category that translates the PB constraints completely into clauses. The DPLL algorithm is used to stay in the valid search space X_f and obtain only feasible solutions. For the first category of PB solvers, the PB constraints are given directly, while for the second category the PB constraints are converted into clauses. The complete algorithm is given in Algorithm 2.

Algorithm 2. Algorithm for multi-objective optimization of Pseudo-Boolean problems based on a DPLL backtracking algorithm

```
1  A = {}
2  while true do
3      branch()
4      status = deduce()
5      if status ==CONFLICT then
6          blevel = analyze_conflict()
7          if blevel < 0 then
8              break
9          else
10             backtrack(blevel)
11         end
12     else if status ==SATISFIABLE ∧ ∀a ∈ A : ¬(a ≽ C^T x) then
13         y = C^T x
14         A = x ∪ {a | a ∈ A ∧ ¬(y ≻ C^T a)}
15     end
16     if ∃ a ∈ A : C^T a ≽ (c_1^T x_1, ..., c_z^T x_z)^T then
17         blevel = 'level of the most recent decision tried not both ways'
18         if blevel < 0 then
19             break
20         else
21             backtrack(blevel)
22         end
23     end
24 end
```

The archive A is holding the set of non-dominated solutions (line 1). The archive is filled and updated throughout the backtracking process until the algorithm aborts and the archive contains the optimal non-dominated solutions, which are the Pareto-optimal solutions.

In a nutshell, line 3 to 11 is identical to a DPLL backtracking algorithm, it ensures that the search process stays in the valid search space X_f: The operation $branch()$ chooses an unassigned variable and assigns it a value. The rules which decide which variable is chosen and value is assigned is called *decision strategy*. The operation $deduce()$ recognizes if any variable assignment is required to keep the constraints satisfiable or a *conflict* occurred. One has to keep in mind that every single constraint has to be satisfied in order to find a feasible solution. Therefore, one decision can cause several necessary assignments, the so called *implications*. If an implication of the same variable occurs to 0 and 1, a conflict is recognized and analyzed in $analyze\_conflict()$ such that a backtracking is triggered. If the backtrack level is less than 0, the first decision was already tested in both ways 0 and 1 and the algorithm is aborted.

In case that all variables have an assignment ($decide()$ returns *SATISFIABLE*) and the current solution is not weakly dominated by any solution inside the archive (line 12), it is added to this archive. At the same time all solutions inside the archive which are weakly dominated by the new solution are removed (line 14).

A backtracking is also triggered if a partial solution is recognized to be weakly dominated by some solutions in the archive independently of its completion. This operation prunes the search space and prevents that the algorithm equals an enumeration of the feasible solutions in X_f. Hence, a lower bound for each objective function has to be calculated and compared for weak domination with the archive (line 16). The lower bounds for the objective functions for a partial solution are calculated separately in each dimension, i.e., a lower bound vector is calculated by $(c_1^T x_1, ..., c_z^T x_z)^T$. Therefore, the vector x_i contains the values of the assigned variables and for unassigned variables a 0 (1) is used if the corresponding coefficient of the vector c_i is positive (negative). The backtracking will take place to the level of the most recent decision that was not tried in both ways 0 and 1 unless this level is lower than 0 what causes an abort of the algorithm (line 17-22).

The used decision strategy is crucial for the success of this algorithm. It is obvious that with good solutions early in the search process and an accurate lower bound calculation large parts of the search space can be pruned. A good approach is a decision strategy that is guided by the coefficients of the objective functions: Focusing on a single-objective problem, variables with a big corresponding coefficient should be favored by the decision strategy to increase the accuracy of the calculated lower bound. This takes place as only variables with small coefficients will be unassigned later in the search process. Moreover, it is desirable to obtain good solutions early in the search process and, as a minimization problem is given, the favored decision phase for a variable with a positive (negative) coefficient should be 0 (1). For multi-objective problems, a more sophisticated decision strategy is needed because variables have different effects in different objective functions. We will propose a static decision strategy based on distribution functions. For each dimension a distribution function $F_i : \mathbb{N} \to [0, 1]$ is approximated by the absolute values of the vector containing the coefficients c_i, thus also a normalization of the coefficients is achieved. We will use a uniform distribution between 0 and the highest value of each dimensions coefficient. For instance, it is also possible to sample the values to a normal or any other distribution. With the given distributions a specific value for each variable can be calculated as follows:

$$\forall i = 1, ..., n : \sum_{j=1}^{z} F_i(|C_{ij}|) sign(C_{ij})$$

According to the rules of the single objective problem the decision strategy uses these values as follows: Variables with a high absolute value calculated by this formula are prioritized in the decision strategy. For a positive (negative) value, the decision takes place to 0 (1). This decision strategy will only work properly if the coefficients are distributed with an adequate variance.

In future work we will extend the algorithm by a dynamic variable order, random restarts, and more precise, but on the other hand slower, lower bound estimation strategies [17].

4.3 Algorithm 3

The third algorithm is an extension of a common iteratively working PB solver. As mentioned before the PB constraints in a PB solver are usually joined by a logical AND.

Instead, the constrained objective function in multi-objective problems have to be joined by logical ORs. To overcome this restriction we will modify the category of specialized PB solvers which translate the PB constraints into clauses and use a common SAT solver to converge to the optimal value. This category of PB solvers is working two-staged as described in [13] and implemented in the PB solver MINISAT+. In the first step each PB constraint is translated into a hardware circuit. In the second step the hardware circuits are translated into a set of clauses. It it obvious that the clauses that are added to a common SAT solver can not be joined by a logical OR as a SAT solver is expecting a conjunctive normal form. As the constrained objective functions need to be joined by a logical OR each time a solution is found, we connect the hardware circuits by ORs and then translate the full circuit to clauses. The complete algorithm is given in Algorithm 3.

Algorithm 3. Algorithm for multi-objective optimization of Pseudo-Boolean problems based on the translation into the Satisfiability problem

```
1  A = {}
2  SATSolver.addClauses(toClauses(toCircuit('Ax ≤ b')))
3  while SATSolver.solve()==SATISFIABLE do
4      x =SATSolver.x()
5      y = C^T x
6      A = x ∪ {a | a ∈ A ∧ ¬(y ≻ C^T a)}
7      h = false
8      foreach i ∈ {1, ..., z} do
9          h = h ∨ toCircuit('c_i^T x < y_i')
10     end
11     SATSolver.addClauses(toClauses(h));
12 end
```

Like in the other algorithms the archive A is holding the set of non-dominated solutions (line 1). It is updated throughout the search process and contains the Pareto-optimal solutions when the algorithm terminates.

Primarily, the PB constraints are translated into clauses (line 2). The translation is two-staged: Each PB constraint is translated into a hardware circuit and afterwards the hardware circuits are translated into clauses. The translation of a PB constraint into a hardware circuit is done by using BDDs, Adders, or Sorters, while the translation of one hardware circuit into a set of clauses is done by using the TSEITIN-transformation, which prevents an exponential number of clauses by introducing additional variables. For a further explanation we strongly recommend [13].

The SAT solver is used iteratively just like in specialized PB solvers to search for non-dominated solutions (line 3). If a non-dominated solution is found, it is added to the archive (line 4-6). Additionally, before the next start of the SAT solver all by this current solution weak dominated solutions have to be excluded from the further search process to guarantee a convergence to the Pareto-optimal solutions. Following Definition 1 of weak dominance this exclusion is done by the formula

$$\neg(f_1(x) \geq y_1 \wedge ... \wedge f_z(x) \geq y_z)$$

for the current found solution y. By using DEMORGAN'S law this can also be interpreted as

$$(f_1(x) < y_1 \vee ... \vee f_z(x) < y_z),$$

which are PB constraints connected by logical ORs. Therefore, the two-staged translation is split as following: The PB constraints are translated into hardware circuits and these circuits are connected by an OR-gate to one hardware circuit (line 7-10). This hardware circuit is then translated into a set of clauses which are added to the SAT solver (line 11).

This approach is showing the high versatility of the category of PB solvers which are translating the PB constraints into SAT. The success of this algorithm depends strongly on the translation of the PB constraints and objective functions into clauses, and the performance of the used SAT solver.

5 Experimental Results

For the experimental results the three algorithms were implemented on the basis of the PB solver MINISAT+ [13] or SAT solver MINISAT [14], respectively, which participated very successful in the past SAT Competition [18] and PB Evaluation [19]. Through using the same PB solver as the basis for all three algorithms, we have the chance for a fair comparison on the basis of runtime. To compare the algorithms, we will use several modifications of the famous queens puzzle and synthetic industrial multi-objective problems from the field of system level synthesis [10]. All test cases were carried out on an Intel Pentium 4 3.20 GHz machine with 1 GB RAM. For each handmade test case 10 instances were created and a representative average was calculated. The timeout bound was set to 1800 seconds.

5.1 Queens Puzzle

Problem Statement. The common queens puzzle is about putting eight queens on a chessboard such that no pair of queens attacks one another. Though a single solution can be obtained by a construction scheme, finding one solution that fulfills the conditions is a non-trivial problem. The problem can easily be converted into a Satisfiability or Pseudo-Boolean problem, respectively, by introducing one variable for each field defining if a queen is located on it or not, and adding the conditions such that in each row and column of the chessboard has to be exactly one queen and in each diagonal at most one queen. Moreover, the queens puzzle can be extended to an $n \times n$ chessboard in which n queens have to be put on this chessboard. The advantage of these handmade problems is that the scaling of the problem size is done by a single variable n instead of many variables, as is the case, e.g., in graph problems where nodes and edges are varied.

To achieve representative test cases we will use the n queens puzzle with appropriate objective functions. The objective functions should have the property to be applicable independently on any number of dimensions. For instance the minimal vertex cover problem can not be scaled to a multi-objective problem. Therefore, we will focus on two optimization classes: the weighted costs optimization and the minimal token optimization. These optimization problems can be extended to any number of dimensions.

Weighted Costs Optimization. Each field of the chessboard gets a cost that is an integer value. The overall costs are a sum of the costs of the fields where a queen is located on, which equals a linear function that has to be minimized. In our examples we will create the single costs randomly as an integer from a uniform distribution between a lower and an upper bound given as integers. The weighted costs optimization problem is denoted as $w(l, h)$ with the lower bound l and the upper bound h. We will analyze two problem classes namely $w(1, 100)$ which we will refer to as *strongly weighted* and $w(0, 1)$ as *weakly weighted*.

Minimal Token Optimization. For each token one variable is introduced. One token exists several times and is distributed randomly on the chessboard. If one field of the chessboard is taken by a queen, the tokens of the field are used which is realized by an implication[2]. The goal is the minimization of the used tokens which is the sum of the variables of the tokens or a linear function with the coefficients 1, respectively. If there are n tokens and one token exists m times and is randomly distributed on the chessboard, we will denote this optimization problem as $t(n, m)$. Instead of some minimization problems where the coefficients are uniquely 1 like the minimal vertex cover, minimal dominating set, or set covering, this problem can be extended to several dimensions by using disjunctive sets of tokens for each dimension.

Analysis of Experimental Results. Several combinations of test cases were carried out and summarized in Table 1. We varied the size, the number of objective functions and the sort of objective function. As one can expect, the problem size and number of objectives affect the runtime of all algorithms. In the strongly weighted problems Algorithm 2 is superior to the other algorithms because an appropriate decision strategy could be calculated. In the weakly weighted problems the decision strategy for Algorithm 2 can not be calculated clearly, which leads to a decline of the runtime. In fact, no algorithm is clearly the best in that problem class. In the minimal token problems Algorithm 3 is superior to the other algorithms. Algorithm 2 is not able to calculate a proper decision strategy as all coefficients are 1 and the algorithm decays to a simple enumeration. Therefore, in all combined problems where at least one objective function is a minimal token optimization Algorithm 2 fails and the best results are provided by Algorithm 3.

5.2 System Level Synthesis

Problem Statement. The task of system level synthesis is to bind a set of communicating processes on a set of interconnected resources and generate feasible *implementations* w.r.t. to a correct communication on the given architecture. Corresponding to the objectives, the goal of design space exploration is to find all optimal implementations which satisfy the specification. For a further explanation we refer to [10]. Searching a single feasible implementation can be formulated as a Satisfiability problem [20]. If the objective functions are linear or linearizable, the resulting problem is a multi-objective Pseudo-Boolean problem.

[2] The field x labeled by a token t leads to an implication $(x \rightarrow t)$, a clause $(\overline{x} \vee t)$, or a PB constraint $x - t \leq 0$, respectively.

Table 1. Results on several queen puzzle problems. Given is the size of the chessboard and the used objective functions. The runtime of the algorithms were calculated as an average each with 10 instances in which the variance is given in the brackets.

Problem	Size	Objective functions	Runtime [s]		
			Algorithm 1	Algorithm 2	Algorithm 3
Queens	10×10	w(1,100)	1.95 (0.49)	**0.28 (0.07)**	2.10 (0.28)
Queens	12×12	w(1,100)	37.9 (7.80)	**5.00 (2.07)**	51.9 (25.5)
Queens	14×14	w(1,100)	1800 (0)	**210 (113)**	1229 (441)
Queens	10×10	w(1,100),w(1,100)	11.3 (4.28)	**0.38 (0.03)**	8.36 (1.14)
Queens	12×12	w(1,100),w(1,100)	699 (176)	**16.7 (2.66)**	227 (25.9)
Queens	14×14	w(1,100),w(1,100)	1800 (0)	**1443 (241)**	1800 (0)
Queens	10×10	w(1,100),w(1,100),w(1,100)	92.3 (22.1)	**0.39 (0.02)**	22.7 (4.18)
Queens	12×12	w(1,100),w(1,100),w(1,100)	1800 (0)	**17.9 (2.76)**	468 (44.8)
Queens	14×14	w(1,100),w(1,100),w(1,100)	1800 (0)	1800 (0)	1800 (0)
Queens	12×12	w(0,1)	0.39 (0.07)	**0.08 (0.01)**	0.09 (0.01)
Queens	14×14	w(0,1)	0.68 (0.13)	**0.12 (0.01)**	0.13 (0.01)
Queens	16×16	w(0,1)	1.05 (0.20)	**0.17 (0.01)**	0.19 (0.01)
Queens	12×12	w(0,1),w(0,1)	1.97 (0.81)	0.79 (0.60)	**0.72 (0.37)**
Queens	14×14	w(0,1),w(0,1)	5.80 (3.71)	8.48 (10.7)	**3.47 (3.87)**
Queens	16×16	w(0,1),w(0,1)	**24.6 (28.3)**	62.5 (124)	35.1 (83.6)
Queens	12×12	w(0,1),w(0,1),w(0,1)	14.5 (7.11)	**6.20 (1.70)**	7.18 (3.45)
Queens	14×14	w(0,1),w(0,1),w(0,1)	**131 (135)**	258 (199)	169 (189)
Queens	16×16	w(0,1),w(0,1),w(0,1)	**1197 (629)**	1800 (0)	1224 (716)
Queens	8×8	t(24,8)	0.23 (0.11)	0.23 (0.8)	**0.05 (0.01)**
Queens	10×10	t(30,10)	0.58 (0.13)	11.5 (5.46)	**0.24 (0.04)**
Queens	12×12	t(36,12)	6.10 (2.59)	1800 (0)	**4.20 (1.18)**
Queens	8×8	t(24,8),t(24,8)	0.87 (0.23)	88.0 (20.8)	**0.08 (0.01)**
Queens	10×10	t(30,10),t(30,10)	3.38 (0.80)	1800 (0)	**0.44 (0.03)**
Queens	12×12	t(36,12),t(36,12)	81.7 (15.1)	1800 (0)	**16.1 (1.66)**
Queens	8×8	t(24,8),t(24,8),t(24,8)	1.98 (0.61)	1800 (0)	**0.14 (0.01)**
Queens	10×10	t(30,10),t(30,10),t(30,10)	15.8 (3.35)	1800 (0)	**0.77 (0.03)**
Queens	12×12	t(36,12),t(36,12),t(36,12)	570 (149)	1800 (0)	**22.3 (2.46)**
Queens	10×10	w(1,100),w(0,1)	3.96 (1.51)	**0.35 (0.05)**	3.52 (0.66)
Queens	12×12	w(1,100),w(0,1)	126 (45.7)	**13.9 (2.56)**	97.5 (20.3)
Queens	10×10	w(1,100),t(30,10)	6.45 (1.96)	1800 (0)	**3.52 (0.70)**
Queens	12×12	w(1,100),t(30,10)	284 (79.4)	1800 (0)	**119 (10.1)**
Queens	10×10	w(0,1),t(30,10)	1.72 (0.66)	1800 (0)	**0.33 (0.03)**
Queens	12×12	w(0,1),t(30,10)	26.2 (8.72)	1800 (0)	**10.2 (1.69)**

The first test case group consists of graphs with 56 processes and 25 resource nodes. For each process the number of mapping edges varies from 3 to 6. This leads to approximately 2^{117} possible solutions. For the second test case group the number of processes was increased to 101 and resources to 50. The mapping edges per process vary from 4 to 8. That leads to about 2^{256} possible solutions. Additionally, the number of feasible solutions was varied from *small* over *medium* to *big*. This is done by specifying the

Table 2. Results on several system level synthesis problems. Given is the number of processes and resources and the number of feasible solutions or size of the valid search space X_f, respectively. The runtime of the algorithms were calculated as an average each with 10 instances in which the variance is given in the brackets.

Problem	Processes	Resources	$\|X_f\|$	Runtime [s]		
				Algorithm 1	Algorithm 2	Algorithm 3
System level Synthesis	56	25	small	226 (332)	**0.20 (0.08)**	4.11 (3.21)
System level Synthesis	56	25	medium	1121 (721)	**0.57 (0.37)**	29.2 (17.8)
System level Synthesis	56	25	big	1800 (0)	**0.94 (1.17)**	55.6 (64.2)
System level Synthesis	101	50	small	1800 (0)	**845 (673)**	1226 (626)
System level Synthesis	101	50	medium	1800 (0)	1800 (0)	1800 (0)
System level Synthesis	101	50	big	1800 (0)	1800 (0)	1800 (0)
H.264 Video Decoder	68	15	-	5.55	**0.05**	1.65

possibility of a connection between two resources. The optimized objectives were the power consumption and the area usage.

Analysis of Experimental Results. The results of the test cases are given in Table 2. As these problems are handling with weighted costs, Algorithm 2 is superior to the other algorithms. The interesting fact is that for a growing number of feasible implementations the complexity is also growing. As typical problems known from real-world applications show hard constrained search spaces containing only a small fraction of feasible solutions [21], this turns out to be advantageous.

H.264 Video Decoder. Concluding we will use an industrial system level synthesis example namely a H.264 Video Decoder. The specification graph contains 68 processes, 15 resources and 276 mapping edges what leads to approximately 2^{136} possible solutions. The proposed algorithms are solving this problem easily, the runtimes are stated in Table 2. Moreover, a multi-objective Evolutionary Algorithm (MOEA) that is usually used to solve these problems will not find the Pareto-optimal solutions even after one hour whereas Algorithm 2 needs just a fraction of one second. The Pareto-optimal solutions and the best solutions of the MOEA after one hour of exploration are illustrated in Figure 2.

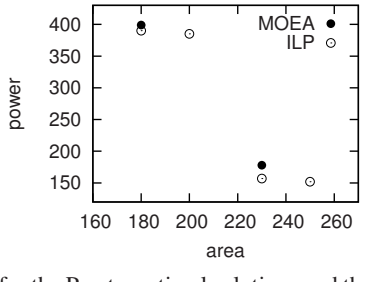


Fig. 2. Pareto-optimal front for the Pareto-optimal solutions and the best non-dominated solutions from a MOEA of the H.264 Video Decoder example. The values area and power are given in abstract units.

6 Conclusions

In this paper we have proposed three algorithms for solving multi-objective Pseudo-Boolean problems. Comparing the algorithms on several examples shows that none of these algorithms is generally superior to the others. Instead of that, the success of the methodologies depends on the given problem and the implementation of the algorithm. The algorithms are, in fact, modifications and extensions of common SAT- and PB solvers, respectively. Therefore, improvements on the field of PB-solving and SAT-solving will consequently also lead to a speed up of the proposed algorithms.

Moreover, we have shown that a typical industrial system level synthesis problem, can be solved in a reasonable amount of time. The example of an H.264 Video Decoder was solved in less than a second what, in this particular case, makes the multi-objective Pseudo-Boolean solvers outstanding in comparison to common multi-objective heuristics like Evolutionary Algorithms.

References

1. Barth, P.: A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Research Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany (1995)
2. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC '71: Proceedings of the third annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (1971) 151–158
3. Karp, R.M.: Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W., eds.: Complexity of Computer Computations. Plenum Press (1972) 85–103
4. Marques-Silva, J.P., Sakallah, K.A.: Boolean satisfiability in electronic design automation. In: DAC '00: Proceedings of the 37th conference on Design automation, New York, NY, USA, ACM Press (2000) 675–680
5. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7) (1962) 394–397
6. Marques-Silva, J.P., Sakallah, K.A.: Grasp - a new search algorithm for satisfiability. In: ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, Washington, DC, USA, IEEE Computer Society (1996) 220–227
7. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient sat solver. In: DAC '01: Proceedings of the 38th conference on Design automation, New York, NY, USA, ACM Press (2001) 530–535
8. Zhang, L., Madigan, C.F., Moskewicz, M.H., Malik, S.: Efficient conflict driven learning in a boolean satisfiability solver. In: ICCAD '01: Proc. of the 2001 IEEE/ACM international conference on Computer-aided design, Piscataway, NJ, USA, IEEE Press (2001) 279–285
9. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Generic ilp versus specialized 0-1 ilp: an update. In: ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design, New York, NY, USA, ACM Press (2002) 450–457
10. Teich, J., Blickle, T., Thiele, L.: An evolutionary approach to system-level synthesis. In: CODES '97: Proceedings of the 5th International Workshop on Hardware/Software Co-Design, Washington, DC, USA, IEEE Computer Society (1997) 167
11. Sheini, H.M., Sakallah, K.A.: Pueblo: A modern pseudo-boolean sat solver. In: DATE '05: Proc. of the conf. on Design, Automation and Test in Europe, Washington, DC, USA, IEEE Computer Society (2005) 684–685

12. Chai, D., Kuehlmann, A.: A fast pseudo-boolean constraint solver. In: DAC '03: Proc. of the 40th conference on Design automation, New York, NY, USA, ACM Press (2003) 830–835
13. Eén, N., Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT. Journal on Satisfiability, Boolean Moelding and Computation **2** (2006) 1–25
14. Eén, N., Sörensson, N.: An extensible sat-solver. In: Conference on Theory and Application of Satisfiability Testing SAT. (2003) 502–518
15. Tseitin, G.: On the Complexity of Derivations in Propositional Calculus. Studies in Contr. Math. and Math. Logic (1968)
16. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evolutionary Computation **7**(2) (2003) 117–132
17. Manquinho, V.M., Marques-Silva, J.: Effective lower bounding techniques for pseudo-boolean optimization. In: DATE '05: Proceedings of the conference on Design, Automation and Test in Europe, Washington, DC, USA, IEEE Computer Society (2005) 660–665
18. Berre, D.L., Simon, L.: Sat competition 2005. Website (2005) Available online at `http://www.satcompetition.org/2005/`.
19. Manquinho, V., Roussel, O.: Pseudo boolean evaluation 2006. Website (2006) Available online at `http://www.cril.univ-artois.fr/PB06/`.
20. Haubelt, C., Teich, J., Feldmann, R., Monien, B.: SAT-Based Techniques in System Design. In Wehn, N., Verkest, D., eds.: Proceedings of Design, Automation and Test in Europe, Munich, Germany, IEEE Computer Society (2003) 1168–1169
21. Deb, K., ed. In: Optimization for Engineering Design. Prentice-Hall of India Pvt.Ltd (1995)

Improved Lower Bounds for Tree-Like Resolution over Linear Inequalities*

Arist Kojevnikov

St.Petersburg Department of Steklov Institute of Mathematics
27 Fontanka, 191023 St.Petersburg, Russia
http://logic.pdmi.ras.ru/ arist/

Abstract. We continue a study initiated by Krajíček of a Resolution-like proof system working with clauses of linear inequalities, R(CP). For all proof systems of this kind Krajíček proved in [1] an exponential lower bound of the form:

$$\frac{exp(n^{\Omega(1)})}{M^{O(W \log^2 n)}} \ ,$$

where M is the maximal absolute value of coefficients in a given proof and W is the maximal clause width.

In this paper we improve this lower bound. For tree-like R(CP)-like proof systems we remove a dependence on the maximal absolute value of coefficients M, hence, we give the answer to an open question from [2]. Proof follows from an upper bound on the real communication complexity of a polyhedra.

Keywords: propositional proof complexity, integer programming, cutting planes.

Many well known methods in an area of pseudo-boolean constraints optimization like a branch-and-bound [3] and Cutting Planes with the deduction rule [4] can be defined in terms of Resolution proof system that operates with clauses of linear inequalities, R(CP) [1]. This proof system is a natural extension of Resolution and can be viewed as a generalization of Resolution over formulas in k-DNF, Res(k), that was introduced in [5]. In the last few years much attention was paid to complexity of Res(k) [6,7,8]. On the other hand, it is not much known about the complexity of R(CP), while it and similar proof systems are often used in practice [9,10,11].

Consider a R(CP)-like proof system as a system that works with clauses of linear inequalities using a finite set of tautologically valid axiom and sound derivation rules with at most two hypotheses. The main goal of this paper is to improve lower bounds on restricted but still very important family of R(CP)-like proof systems. Namely, we proved better lower bounds for tree-like R(CP)-like proof systems.

* Supported in part by Russian Science Support Foundation, INTAS (grant 04-83-3836) and RFBR (grants 05-01-00932, 06-01-00502). The paper was done during the stay of the author at the Max-Planck-Institut für Mathematik, Bonn, Germany.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 70–79, 2007.

The main idea of exponential lower bounds that are based on monotone interpolation theorems is a transformation of a proof P of the formula F into a monotone circuit C of size polynomial in $|P|$. If the formula F formalizes that the intersection of two disjoint NP-sets is not empty, then the circuit C separates these two disjoint NP-sets. For example, the pair of disjoint NP-sets, consisting of a set of graphs with a k-clique and the set of $(k-1)$-colorable graphs, the monotone circuit that separates one set from another has at least exponential size [12]. Hence, the size of proof P is exponential.

There is a very nice connection between boolean circuits and communication complexity [13], and sometimes it is easier to think in terms of communication complexity then in terms of circuits. This idea was used by Krajíček to prove many important exponential lower bounds in [14,1,2]. He reduced the proof-into-circuit transformation problem into a problem of proving upper bounds on communication complexity of specific decision problems.

In this paper we give an answer to one of the open questions from [2]: we prove new upper bound on real monotone communication complexity of a polyhedra and, hence, a better lower bound for tree-like R(CP)-like proof systems. The proof is straightforward. The basic techniques are the same as in [14,1,2].

The paper is organized as follows. In Sect. 1 we give all necessary definitions, in Sect. 2 we recall the notion of interpolation and prove new lower bound on tree-like R(CP)-like proof systems. In Sect. 3 we discuss related open questions.

1 Definitions

In this paper we use the following notation: we typically denote integer vectors with letters a, b, c, their coordinates with a_i, b_i, c_i, vectors of Boolean variables with u, v, w, x, y, z and integers with A, B, C. We will write $a \cdot x$ instead of $\sum_i a_i x_i$.

1.1 Resolution over Linear Inequalities

Now we describe several propositional proof systems for the language of systems of linear inequalities that have no 0/1-solutions. A proof system R(CP) was defined in [1] as follows. The lines of the system are disjunctions of linear inequalities: $a \cdot x \geq A \vee \ldots \vee b \cdot x \geq B$. The derivation rules are (we denote by Γ an arbitrary disjunction of linear inequalities)

$$\frac{a \cdot x \geq A \vee \Gamma \qquad b \cdot x \geq B \vee \Gamma}{(a+b) \cdot x \geq A + B \vee \Gamma}\ , \qquad \frac{a \cdot x \geq A \vee \Gamma}{Ca \cdot x \geq CA \vee \Gamma}\ , \quad \text{where } C \geq 0\ ,$$

$$\frac{Ca \cdot x \geq A \vee \Gamma}{a \cdot x \geq \lceil A/C \rceil \vee \Gamma}\ , \qquad \frac{}{x_i \geq 0} \qquad \frac{}{-x_i \geq -1} \quad \text{for all variables } x_i\ ,$$

$$\frac{}{a \cdot x \geq A \vee (-a) \cdot x \geq 1 - A}\ , \quad \frac{\Gamma}{a \cdot x \geq A \vee \Gamma}\ , \quad \frac{a \cdot x \geq A \vee a \cdot x \geq A \vee \Gamma}{a \cdot x \geq A \vee \Gamma}\ .$$

Note that one can omit $0 \geq 1$ from $0 \geq 1 \vee \Gamma$ because the contradiction $0 \geq 1$ is easily transformable into any other inequality. The goal is to derive $0 \geq 1$.

We also define a family of R(CP)-like proof systems, that operate with disjunctions of linear inequalities by finite set of tautologically valid axioms and sound derivation rules that have at most two hypotheses. We are interested in its sub-family of *p-passive* R(CP)-like proof systems, where all derivation rules are of the form

$$\frac{\Delta_1 \vee \Gamma_1 \qquad \Delta_2 \vee \Gamma_2}{\Delta_3 \vee \Gamma_1 \vee \Gamma_2} \ ,$$

where Δ_i and Γ_i are arbitrary disjunctions of linear inequalities and $|\Delta_i| \leq p$, for $i = 1, 2$.

1.2 Real Communication Complexity

The following set of definitions is an extension of boolean communication complexity [13,15], that allows players to communicate with each other not only by bits, but with real numbers. It was introduced in [2].

Let I be finite set, $U, V \subset \{0,1\}^*$, $R \subseteq U \times V \times I$ be such that

$$\forall u \in U,\ v \in V\ \exists i \in I\ R(u,v,i) \ .$$

We will call relations satisfying this condition *multifunctions*.

The following two definitions were given in [16].

Definition 1. *A* real communication protocol *P over $U \times V$ with range I is a binary tree where each internal node v is labeled by two functions $a_v : U \to \mathbb{R}$, giving player A move, and $b_v : V \to \mathbb{R}$, giving player B move, and each leaf is labeled by an element $i \in I$.*

On input (x, y), the players construct a path thought the tree according to the following rule: At each internal node v labeled by (a_v, b_v), if $a_v(x) > b_v(y)$, then the next node is the left son of v and otherwise the right son of v. If for every $u \in U$ and $v \in V$ the value i of P satisfies $R(u,v,i)$, we say that P computes *R.*

Definition 2. *The* real communication complexity *of a multifunction R, $CC^{\mathbb{R}}(R)$, is the minimal depth of a real communication protocol P, over all P that compute R.*

Usually, sets U, V are defined by some partial Boolean function f that maps $W \subseteq \{0,1\}^n$ to $\{0,1\}$. We take $U := f^{-1}(1)$, $V := f^{-1}(0)$ and $I := \{1, \ldots, n\}$. Relation $R(u,v,i)$ is true if strings u and v differ in position i. We are interested in *monotone* partial Boolean functions, that have at least one extension to a monotone Boolean function [13]. For such a function f define $R_f^{mono} \subseteq U \times V \times I$ by

$$R_f^{mono}(u,v,i) \quad \text{iff} \quad u \in U \wedge v \in V \wedge u_i = 1 \wedge v_i = 0 \ .$$

As it happens with monotone boolean functions and Boolean communication complexity, there is a relation between the real communication complexity of R_f^{mono} and the depth of monotone real circuit computing f.

1.3 Monotone Real Circuits

A *monotone real circuit* is a circuit of fan-in 2 computing with real numbers where every gate computes a nondecreasing real function [17]. Since monotone real circuits are generalization of monotone boolean circuits, we require that they output 0 or 1 on every input from $\{0,1\}^*$. The depth and size of the monotone real circuit are defined as for boolean circuits.

Lemma 1 (Lemma 1.4, [2]). *Let f be a partial monotone boolean function. Then $CC^{\mathbb{R}}(R_f^{mono})$ is at most the minimal depth of a monotone real circuit C that computes the function f. Moreover,*

$$CC^{\mathbb{R}}(R_f^{mono}) \leq \log_{3/2} S^{\mathbb{R}}(f) \ ,$$

where $S^{\mathbb{R}}(f)$ is the minimal size of a monotone real formula computing f.

There is an important open question about the converse statement. A positive answer on it would immediately imply an extension of lower bound proved in this paper from tree-like R(CP) to general R(CP) [2].

1.4 Local Search Protocols

The notions of *local search protocol* and *monotone local search protocol* were defined in [14] and they generalize the notion of real communication protocol. We need them for transformation of a refutation in some proof system into the real circuit in a natural and intuitive way.

Definition 3 (Definition 2.1, [2]). *Let $U, V \subseteq \{0,1\}^n$ be two sets and let $R \subseteq U \times V \times I$ be a multifunction. A* local search protocol *for R is a labeled directed graph G satisfying the following conditions:*

1. *Graph G is acyclic and has one source denoted by $\emptyset$. The nodes with zero out-degree are leaves, all other are inner nodes. All inner nodes have out-degree 2.*
2. *All leaves are labeled by elements of I.*
3. *There is a* strategy *$S(u,v,x)$ that assigns to a node x and a pair $u \in U$ and $v \in V$ one of the two children $S(u,v,x)$ of x.*
4. *For every pair $u \in U$, $v \in V$ there is a set $F(u,v)$ of nodes of G satisfying:*
   (a) $\emptyset \in F(u,v)$.
   (b) $x \in F(u,v) \rightarrow S(u,v,x) \in F(u,v)$.
   (c) If i is the label of a leaf from $F(u,v)$ then $R(u,v,i)$ holds.
   We call such set F the consistency condition.

The local search protocol is tree-like *iff the underlying graph is a tree.*

A local search protocol for a particular multifunction $R = \{(u,v,i) | u_i = 1 \wedge v_i = 0\}$ is called a monotone local search protocol *for U, V.*

Definition 4 (Definition 2.2, [2]). *Let G be a local search protocol for R. Let $S(u,v,x)$ be the strategy and $F(u,v)$ be the consistency condition of G.*

The real communication complexity of G, *denoted $CC^{\mathbb{R}}(G)$, is the minimal t such that for every $x \in G$ the players (first knows pair (u,x), the second knows (v,x)) decide $x \in F(u,v)$ and compute $S(u,v,x)$ by real communication protocol by depth at most t.*

For tree-like local search protocol it is possible to prove an exponential lower bounds on the following set of functions:

Let I, J be sets of size n. Consider a monotone Boolean function BPM that gives to a bipartite graph $\Gamma \subseteq I \times J$ the value 1 iff Γ contains a perfect matching. Inputs to BPM are n^2 variables x_{ij}, $i \in I, j \in J$. Their truth evaluations are in one to one correspondence with bipartite graphs.

Theorem 1 (Theorem 2.5, [2]). *Let G be a tree-like local search protocol for BPM of size S, such that $CC^{\mathbb{R}}(G) = t$. Then*

$$S = exp(\Omega((\frac{n}{t \log n})^{1/2})) \ .$$

2 Lower Bound for Tree-Like R(CP)-Like Proof Systems

The following definition was introduced in [14] and is a generalization of usual derivation in a proof system. A sequence of sets $D_1, \ldots, D_k \subseteq \{0,1\}^N$ is a *semantic derivation* of D_k from $A_1, \ldots, A_m$ if each D_i is either one of A_j, or contains $D_{i_1} \cap D_{i_2}$ for some $i_1, i_2 < i$. Till the end of this section we use $N = n + s + t$. Let us consider the following problem for two players:

Definition 5 (Definition 3.1, [2]). *For set $A \subseteq \{0,1\}^N$ we fix $u, v \in \{0,1\}^n$, $y \in \{0,1\}^s$ and $z \in \{0,1\}^t$. Consider the following three tasks:*

1. *Decide whether $(u,y,z) \in A$.*
2. *Decide whether $(v,y,z) \in A$.*
3. *If $(u,y,z) \in A$ and $(v,y,z) \notin A$, then find such $i \leq n$ that*

$$u_i = 1 \wedge v_i = 0$$

or find some u' satisfying

$$u' \geq u \wedge (u',y,z) \notin A \qquad (\text{where } u' \geq u \text{ means } \textstyle\bigwedge_{i \leq n}(u'_i \geq u_i)) \ .$$

These tasks can be solved by two players, one knowing (u,y) and another one knowing (v,z).

A monotone real communication complexity *of A, $MCC^{\mathbb{R}}(A)$ is the minimal t such that tasks 1-3 have real communication complexity at most t.*

We define subset $Q(b)$ of $\mathbb{Z}^W$ as follows

$$Q(b) = \{a \in \mathbb{Z}^W \,|\, \forall i \leq W \ (a_i \leq b_i - 1)\} \ .$$

We need to prove the following lemma to improve the lower bound for tree-like R(CP)-like proof systems. It extends Lemma 5.1, [14] to real communication complexity.

Lemma 2. *Let linear mapping*

$$H : \{0,1\}^N \to \mathbb{Z}^W$$

be defined by a matrix with elements from $\mathbb{Z}$.
Let $Y \subseteq \mathbb{Z}^W$ *be any set defined as*

$$Y = \mathbb{Z}^W \setminus Q(b) \ ,$$

for some $b \in \mathbb{Z}^W$. *We fix* $X := H^{-1}(Y)$.
Then

$$MCC^{\mathbb{R}}(X) = O(W) + O(\log(n)) \ .$$

Proof. 1. To decide whether $(u,y,z) \in X$ we need to find such $i \in 1,...,W$ that

$$\sum_{j=1}^{n} h_{ij} \cdot u_j + \sum_{j=n+1}^{n+s} h_{ij} \cdot y_j + \sum_{j=n+s+1}^{n+s+t} h_{ij} \cdot z_j \geq b_i \ . \tag{1}$$

Player A knows all elements in this sum except z. Let integer z_i satisfy the equality

$$\sum_{j=1}^{n} h_{ij} \cdot u_j + \sum_{j=n+1}^{n+s} h_{ij} \cdot y_j + z_i = b_i \ .$$

The players compare z_i and $z'_i = \sum_{j=n+s+1}^{n+s+t} h_{ij} \cdot z_j$ for all $i \in 1,...,W$ and if for some i the inequality $z_i \leq z'_i$ holds, then (1) also holds and therefore $(u,y,z) \in X$. Otherwise, $(u,y,z) \notin X$.

To decide whether $(u,y,z) \in X$ players use the real communication protocol of depth W.

2. Similarly, by real communication protocol of depth W, players can decide whether $(v,y,z) \in X$.
3. Assume that $(u,y,z) \in X$ and $(v,y,z) \notin X$. It means that for some $i \in 1,...,W$ is

$$\sum_{j=1}^{n} h_{ij} \cdot u_j + \sum_{j=n+1}^{n+s} h_{ij} \cdot y_j + \sum_{j=n+s+1}^{n+s+t} h_{ij} \cdot z_j \geq b_i \ ,$$

and also

$$\sum_{j=1}^{n} h_{ij} \cdot v_j + \sum_{j=n+1}^{n+s} h_{ij} \cdot y_j + \sum_{j=n+s+1}^{n+s+t} h_{ij} \cdot z_j < b_i \ .$$

From the last two inequalities it follows that

$$\sum_{j\in J} h_{ij}\cdot u_j > \sum_{j\in J} h_{ij}\cdot v_j \ ,$$

where $J=\{1,\dots,n\}$.

For all j such that $h_{ij}<0$ first player assigns 1 to u_j. If for some $u'\geq u$ the triple $(u',y,z)\notin X$, then he communicates one bit of the answer to second player, and they stop if it is equal to 1. Otherwise,

$$\sum_{j\in J} h_{ij}\cdot u'_j > \sum_{j\in J} h_{ij}\cdot v_j \ , \tag{2}$$

where $J=\{1,\dots,n\}$.

Let fix $J_1=\{1,\dots,\lfloor n/2\rfloor\}$ and $J_2=\{\lfloor n/2\rfloor+1,\dots,n\}$. Note that it holds either

$$\sum_{j\in J_1} h_{ij}\cdot u'_j > \sum_{j\in J_1} h_{ij}\cdot v_j \qquad \text{or} \qquad \sum_{j\in J_2} h_{ij}\cdot u'_j > \sum_{j\in J_2} h_{ij}\cdot v_j \ ,$$

otherwise (2) is not satisfying. Continue with one of the satisfied inequalities and find such j that $(u'_j=1\wedge v_j=0)$ or $(u'_j=0\wedge v_j=1)$. Since in this case, $h_{ij}>0$ (otherwise u'_j is equal to 1), we have that $u'_j=u_j=1\wedge v_j=0$.

The real communication complexity of described binary search procedure is equal to $O(\log(n))$. □

Following [2] we define a set $\tilde{A}$ for the $A\subseteq\{0,1\}^{n+s}$ as follows:

$$\tilde{A} := \bigcup_{(a,b)\in A} \{(a,b,c)\mid c\in\{0,1\}^t\} \ ,$$

where a,b,c are from $\{0,1\}^n,\{0,1\}^s$ and $\{0,1\}^t$ respectively. For $B\subseteq\{0,1\}^{n+t}$ we define in the same way $\tilde{B}$:

$$\tilde{B} := \bigcup_{(a,c)\in B} \{(a,b,c)\mid b\in\{0,1\}^s\} \ .$$

Theorem 2 (Theorem 3.2, [2]). *Let $A_1,\dots,A_m\subseteq\{0,1\}^{n+s}$ and $B_1,\dots,B_\ell\subseteq\{0,1\}^{n+t}$ be two set families. Assume that there is a semantic derivation $\pi=D_1,\dots,D_k$ of the empty set $\emptyset=D_k$ from $A_1,\dots,A_m,B_1,\dots,B_\ell$. Assume also that all the sets $A_1,\dots,A_m$ satisfy the following monotone condition:*

$$(u,y)\in\bigcap_{j\leq m} A_j \wedge u\leq u' \to (u',y)\in\bigcap_{j\leq m} A_j$$

and $MCC^{\mathbb{R}}(D_i)\leq t$ for all $i\leq k$.

Define sets U and V as follows:

$$U = \{u \in \{0,1\}^n \mid \exists y \in \{0,1\}^s; (u,y) \in \bigcap_{j \le m} A_j\}$$

and

$$V = \{v \in \{0,1\}^n \mid \exists z \in \{0,1\}^t; (v,z) \in \bigcap_{j \le \ell} B_j\} \ .$$

Then there is a monotone local search protocol G for the sets U, V of size at most $k+n$ with real communication complexity $CC^{\mathbb{R}}$ at most t.

Moreover, if the semantic derivation π is tree-like, then local search protocol G is also tree-like.

The following theorem extends [2, Theorem 3.3] from CP-like proof systems to R(CP)-like proof systems.

Theorem 3. *Let a system of linear inequalities $E_1(x,y)$, ..., $E_m(x,y)$, $F_1(x,z)$, ..., $F_\ell(x,z)$ contain only variables $(x_1,\ldots,x_n)$, $(y_1,\ldots,y_s)$ and $(z_1,\ldots,z_t)$. Assume that there is a refutation π of the system in* R(CP)*-like proof system with k lines. Let every clause in π have at most W occurrences of linear inequalities. Assume also that x_i occur in all $E_1,\ldots,E_m$ only with non-negative coefficients.*

Then there is a monotone local search protocol G for U, V:

$$U = \{u \in \{0,1\}^n \mid \exists y \in \{0,1\}^s; (u,y) \text{ satisfying } \bigwedge_{i \le m} E_i(u,y)\} \ ,$$

$$V = \{v \in \{0,1\}^n \mid \exists z \in \{0,1\}^t; (v,z) \text{ satisfying } \bigwedge_{j \le \ell} F_j(v,z)\} \ ,$$

such that the size of G is at most $k+n$ and its real communication complexity is $O(W) + O(\log(n))$.

Moreover, if the refutation π is tree-like, then local search protocol G is also tree-like.

Proof. Consider a clause $D = \{h_i \cdot (x,y,z)^T \ge b_i \mid i \le W\}$ in the refutation π. Then assignment (x,y,z) satisfies it iff

$$H \cdot (x,y,z) \in \mathbb{Z}^W \setminus Q((b_1,\ldots,b_W)) \ ,$$

where H is a $N \times W$-matrix with strings h_i. Replace each clause D in π by $\tilde{D} \subseteq \{0,1\}^N$ of assignments satisfying it to obtain a semantic refutation of $\tilde{E}_i$ and $\tilde{F}_i$. By Lemma 2 for every set S occurring in the refutation it holds that $MCC^{\mathbb{R}}(S) = O(W) + O(\log(n))$. To complete the proof apply Theorem 2. □

2.1 Exponential Lower Bounds

In [2] the following set of inequalities was introduced, $Hall_n$, that formalize Hall's theorem.

Let $|I| = |J| = n$.

1. $\sum_i y_{ki} \geq 1$, for all $1 \leq k \leq n$.
2. $y_{ki} + y_{k'i} \leq 1$, for all $1 \leq k < k' \leq n$.
3. $\sum_j y'_{kj} \geq 1$, for all $1 \leq k \leq n$.
4. $y'_{kj} + y'_{k'j} \leq 1$, for all $1 \leq k < k' \leq n$.
5. $y'_{kj} + y_{ki} - x_{ij} \leq 1$, for all $1 \leq k \leq n$, $i \in I$, $j \in J$.

Let $E_i(x, y, y')$ be all these linear inequalities. Note, that the set

$$U := \{x \in \{0,1\}^{n^2} \mid \exists y, y'(\bigwedge_i E_i(x, y, y'))\}$$

determines a set of graphs with BPM equal to 1.

The set V of graphs with BPM equal to 0 can be defined analogously by inequality system $F_j(x, z, z')$. The union set of all inequalities E_i and F_j is denoted by $Hall_n$.

Theorem 4. *Let π be a tree-like refutation of $Hall_n$ in any* R(CP)*-like proof system. Then $|\pi| \geq exp(\Omega((\frac{n}{W\log(n)+(\log(n))^2})^{1/2}))$.*

Proof. By Theorem 3 there is a tree-like monotone local search protocol G for BPM problem of size $k + n$ and real communication complexity $t = O(W) + O(\log(n))$. The required lower bound follows from Theorem 1. □

3 Open Questions

In this section we formulate some important open questions.

1. Remove a dependence on maximal absolute value of coefficient in Krajíček's exponential lower bound for general R(CP)-like proof system or prove that it is impossible.
2. Remove a dependence on maximal number of inequalities in clauses for tree-like R(CP)-like proof systems or prove that it is impossible.

Remark 1 (Iddo Tzameret). We cannot remove a dependence on the maximal number of inequalities in clauses for general R(CP)-like proof systems, since in [18] it was proved that Res(2) (a subsystem of R(CP) with polynomially bounded coefficients) does not have monotone interpolation in almost exponential time.

Acknowledgments

The author is very grateful to Dima Grigoriev, Jan Krajíček, Alexander S. Kulikov and Iddo Tzameret for helpful comments and is indebted to Edward A. Hirsch for enlightening discussions.

References

1. Krajíček, J.: Discretely ordered modules as a first-order extension of the cutting planes proof system. Journal of Symbolic Logic **63**(4) (1998) 1582–1596
2. Krajíček, J.: Interpolation by a game. Mathematical Logic Quarterly **44**(40) (1998) 450–458

3. Land, H., Doig, A.G.: An automatic method for solving discrete programming problems. Econometrica **28** (1960) 497–520
4. Bonet, M., Pitassi, T., Raz, R.: Lower bounds for cutting planes proofs with small coefficients. The Journal of Symbolic Logic **62**(3) (1997) 708–728
5. Krajíček, J.: On the weak pigeonhole principle. Fundamenta Mathematicæ**170** (1-3) (2001) 123–140
6. Atserias, A., Bonet, M.L., Esteban, J.L.: Lower bounds for the weak pigeonhole principle and random formulas beyond resolution. Information and Computation **176**(2) (2002) 136–152
7. Segerlind, N., Buss, S.R., Impagliazzo, R.: A Switching Lemma for Small Restrictions and Lower Bounds for k-DNF Resolution. SIAM Journal on Computing **33**(5) (2004) 1171–1200
8. Alekhnovich, M.: Lower bounds for k-DNF resolution on random 3-CNFs. In: STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (2005) 251–256
9. Prestwich, S.: Incomplete dynamic backtracking for linear pseudo-boolean problems. Annals of Operations Research **130** (2004) 57–73
10. Chai, D., Kuehlmann, A.: A fast pseudo-boolean constraint solver. IEEE Trans. on CAD of Integrated Circuits and Systems **24**(3) (2005) 305–317
11. Manquinho, V.M., Marques-Silva, J.: On using cutting planes in pseudo-boolean optimization. Journal of Satisfiability, Boolean Modeling and Computation **2** (2006) 209–219
12. Razborov, A.A.: Lower bounds on the monotone complexity of some Boolean functions. Dokl. Akad. Nauk SSSR **281**(4) (1985) 798–801 In Russian: English translation in *Soviet Math. Dokl.* 31:354–357, 1985.
13. Karchmer, M., Wigderson, A.: Monotone circuits for connectivity require superlogarithmic depth. SIAM Journal on Discrete Mathematics **3**(2) (1990) 255–265
14. Krajíček, J.: Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. Journal of Symbolic Logic **62**(2) (1997) 457–486
15. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press (1997)
16. Bonet, M.L., Esteban, J.L., Galesi, N., Johannsen, J.: On the relative complexity of resolution refinements and cutting planes proof systems. SIAM J. Comp. **30**(5) (2000) 1462–1484
17. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. Journal of Symbolic Logic **62**(3) (1997) 981–998
18. Atserias, A., Bonet, M.L.: On the automatizability of resolution and related propositional proof systems. Information and Computation **189**(2) (2004) 182–201

Horn Upper Bounds and Renaming*

Marina Langlois, Robert H. Sloan, and György Turán**

University of Illinois at Chicago
Chicago, IL 60607, USA
{mirodo1, sloan, gyt}@uic.edu

Abstract. We consider the problem of computing tractable approximations to CNF formulas, extending the approach of Selman and Kautz to compute the Horn-LUB to involve renaming of variables. Negative results are given for the quality of approximation in this extended version. On the other hand, experiments for random 3-CNF show that the new algorithms improve both running time and approximation quality. The output sizes and approximation errors exhibit a 'Horn bump' phenomenon: unimodal patterns are observed with maxima in some intermediate range of densities. We also present the results of experiments generating pseudo-random satisfying assignments for Horn formulas.

1 Introduction

A general formulation of the reasoning problem in propositional logic is to decide if a clause C is implied by a CNF expression φ. Here φ is often viewed as a fixed *knowledge base*, and it is assumed that a large number of *queries* C have to be answered for the same knowledge base. Therefore, it may be useful to preprocess φ into a more tractable form, resulting in a new knowledge base which may be only approximately equivalent to the original one. This approach, called *knowledge compilation*, goes back to the seminal work of Selman and Kautz [22] (see also [5, 7, 24]).

Selman and Kautz suggested considering *Horn formulas* approximating the initial knowledge base from above and below, and using these formulas to answer the queries. In particular, they gave an algorithm (outlined in Section 4) computing a *Horn least upper bound (Horn-LUB)* of φ which is equivalent to the conjunction of all its Horn prime implicates.[1] The set of truth assignments satisfying the Horn-LUB of φ has a natural combinatorial characterization which suggests that this notion may be of interest in itself. The intersection of two truth assignments is obtained by taking their componentwise conjunction. In

* This material is based upon work supported by the National Science Foundation under grant CCF-0431059. A preliminary version of this work appeared in [14].

** Also affiliated with Hungarian Academy of Sciences and University of Szeged, Research Group on Artificial Intelligence, Szeged, Hungary.

[1] We omit the definition of Horn *greatest lower bounds*, as those will not be discussed in this paper.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 80–93, 2007.

other words, the intersection is the greatest lower bound of the two truth assignments in the componentwise partial ordering of the hypercube. The set of truth assignments satisfying the Horn-LUB of φ, then, can be obtained as the *closure under intersections of the set of satisfying truth assignments of* φ.

Queries to Horn formulas can be answered efficiently, but the approach can have the following drawbacks: it may be inefficient as the resulting Horn upper bound may be large, and it may fail to answer certain queries (those implied by the lower bound, but not implied by the upper bound). Indeed, such theoretical negative results on the worst-case performance of the approach have been obtained in Selman and Kautz [22] and del Val [8]. Another interesting question is the performance of the algorithms on random examples. Initial experiments in this direction were performed by Kautz and Selman [12]. Boufkhad [4] gives the results of experiments for Horn lower bounds using an extension to renamable Horn formulas. The main test examples were random 3-CNF formulas with density around 4.2, which are well known to be hard for satisfiability algorithms.[2] In other related work, Van Maaren and van Norden [25] considered the connection between the efficiency of satisfiability algorithms and the size of a largest renamable sub-CNF for random 3-CNF.

In this paper we introduce new variants of the Horn-LUB algorithm and present theoretical and experimental results on their performance. The new variants involve the construction of a renaming of some variables, i.e., switching some variables and their complements. Thereby one hopes to bring the original formula closer to being a Horn formula, possibly resulting in a smaller Horn-LUB with better approximation quality. We use an algorithm of Boros [3] to find a large renamable subformula of the original knowledge base. Another possibility we consider is the use of resolvents of bounded size only. This is expected to speed up the algorithm and decrease the size of the Horn upper bound at the price of decreasing approximation quality. Thus it may be of interest to explore the trade-offs to find an optimal size bound. The combinatorial interpretation of the Horn-LUB mentioned above carries over to the case of renaming. Informally, a renaming corresponds to a reorientation of the hypercube, by choosing an arbitrary vector as the 'bottom' of the hypercube instead of the all 0's vector. In order to obtain the Horn-LUB after the renaming, intersections have to be taken with respect to this new orientation.

The theoretical results show similar worst-case behavior as in the original case. In particular, 3-CNF expressions are presented with only a polynomial number of truth assignments such that for every renaming of the variables, the Horn-LUB obtained after the renaming has superpolynomially many satisfying truth assignments. We also construct a polynomial size CNF expression such that for every renaming, only a superpolynomially small fraction of the prime implicates are Horn, and therefore most of the prime implicate queries are answered incorrectly by the renamed Horn-LUB.

In the second part of the paper we present experiments indicating that, on the other hand, the new algorithms give improvements in both efficiency and

[2] Kautz and Selman also considered a class of planning problems, and Boufkhad also considered 4-CNF formulas.

approximation quality. We have compared Selman and Kautz's original algorithm and three variants. The best one appears to be the one that uses both renaming and bounded size resolvents. As the performance of the algorithms is evaluated by exhaustive testing over all truth assignments, we have chosen to run the experiments for 20 variables. In order to consider a larger number of variables, it would be necessary to be able to efficiently sample random satisfying truth assignments of a Horn formula. It appears to be an open question whether this is possible. We have run experiments with some natural candidates for such a sampling algorithm. The results of these experiments are also included in the paper.

In contrast to previous work, in the experiments we have considered 3-CNF formulas of *different densities*, in particular, for densities well below the critical range. Here we have observed an interesting phenomenon—the *Horn bump*: the performance of each algorithm is the worst in an intermediate range of densities. This phenomenon may be of interest for the study of the evolution of random 3-CNF formulas [19, 18, 21].

2 Preliminaries

A clause is a disjunction of literals; a clause is Horn (resp., definite, negative) if it contains at most one (resp., exactly one, no) unnegated literal. A CNF is a conjunction of clauses; it is a 3-CNF if each clause contains exactly 3 literals. A clause C is an *implicate* of a CNF expression φ if every *truth assignment* or *vector* in $\{0,1\}^n$ satisfying φ also satisfies C; it is a *prime* implicate if none of its sub-clauses is an implicate. An n-variable *random 3-CNF* formula of *density* α is obtained by selecting $\alpha \cdot n$ clauses of size 3, selecting each clause from the uniform distribution over all such clauses. A *Horn formula* or *Horn-CNF* is a conjunction of Horn clauses.

The set of satisfying truth assignments of a formula φ is denoted by $T(\varphi)$. The weight of a 0-1 vector is number of its 1 components. The *intersection* of vectors $(x_1, \ldots, x_n)$, $(y_1, \ldots, y_n) \in \{0,1\}^n$ is $(x_1 \wedge y_1, \ldots, x_n \wedge y_n)$. A Boolean function can be described by a Horn formula if and only if its set of satisfying truth assignments is closed under intersection [10, 17]. The *(ordinary) Horn closure* $\mathcal{H}(S)$ of any set S of truth assignments is the smallest intersection-closed set of truth assignments containing S.

A Horn least upper bound of φ (Horn-LUB(φ)) is any conjunction of Horn clauses logically equivalent to the conjunction of all Horn prime implicates of φ. The set of satisfying truth assignments of Horn-LUB(φ) is $\mathcal{H}(T(\varphi))$, the Horn closure of $T(\varphi)$. Selman and Kautz [22] give an algorithm for computing a Horn-LUB(φ).

Renaming a variable x in a CNF is the operation of simultaneously *switching* every occurrence of x to $\bar{x}$ and of $\bar{x}$ to x. A *renaming (function) with respect to vector* $d \in \{0,1\}^n$, denoted by $\mathcal{R}_d$, maps a CNF formula φ to $\mathcal{R}_d(\varphi)$, obtained by switching every pair of literals x_i and $\bar{x}_i$ such that $d_i = 1$. The following easily verified proposition shows that the operation on truth assignments corresponding

to renaming w.r.t. vector d is taking the exclusive or with d, and one can use renaming to solve the reasoning problem formulated in the introduction.

Proposition 1. a) *A truth assignment a satisfies CNF φ iff the truth assignment $a \oplus d$ satisfies $\mathcal{R}_d(\varphi)$.*

b) *For any CNF φ, clause C, and vector d, we have $\varphi \models C$ if and only if $\mathcal{R}_d(\varphi) \models \mathcal{R}_d(C)$.*

A CNF φ is *Horn renamable* if $\mathcal{R}_d(\varphi)$ is a Horn formula for some vector d. It can be decided in polynomial time if a CNF is Horn renamable [1,16], but finding a largest Horn renamable sub-CNF of a given CNF is NP-hard [6]. Boros [3] gave an approximation algorithm for finding a large Horn renamable sub-CNF in an arbitrary CNF in linear time. Given a direction $d \in \{0,1\}^n$, the *d-Horn closure of a set S of truth assignments* is

$$\mathcal{H}_d(S) = \mathcal{H}(\{a \oplus d : a \in S\}).$$

With an abuse of notation, we refer to $\mathcal{H}_d(T(\varphi)) = \mathcal{H}(T(\mathcal{R}_d(\varphi)))$ as the *d-Horn closure* of φ.

3 Negative Results for Horn Upper Bounds with Renaming

In this section we present negative results for Horn closures and Horn-LUB with renaming, analogous to those for the ordinary Horn closure and Horn-LUB.

Theorem 1. *There are 3-CNF formulas φ with a polynomial number of satisfying truth assignments such that for every direction d, the size of the d-Horn closure of φ is superpolynomial.*

Proof. The construction uses the following lemma.

Lemma 1. *There is a set $S \subseteq \{0,1\}^m$ with $|S| = 2m$ such that for every direction $d \in \{0,1\}^m$ it holds that*

$$|\mathcal{H}_d(S)| \geq 2^{\lceil m/2 \rceil}.$$

Proof. Let S be the set of vectors of weight 1 and $(m-1)$ and let $d \in \{0,1\}^m$ be any direction. Then d has at least $\lceil m/2 \rceil$ 0's or $\lceil m/2 \rceil$ 1's. Assume w.l.o.g. that the first $\lceil m/2 \rceil$ components of d are 0 (resp., 1). Consider those vectors from S which have a single 0 (resp. 1) in one of the first $\lceil m/2 \rceil$ components. All possible intersections of these vectors (resp., the complements of these vectors) are contained in the d-Horn closure of S. Thus all possible vectors on the first $\lceil m/2 \rceil$ components occur in the d-Horn closure and the bound of the lemma follows. □

Now consider the 4-CNF φ formed by taking the conjunction of all possible clauses of size 4 containing two unnegated and two negated literals over m variables. The vectors satisfying this formula are those in the set S in the proof of

Lemma 1 plus the all 0's and the all 1's vectors. Hence by Lemma 1, φ's Horn closure with respect to any direction has size at least $2^{\lceil m/2 \rceil}$.

In order to obtain a 3-CNF ψ, introduce a new variable z for each clause $(a \vee b \vee c \vee d)$ in φ, and replace the clause by five new clauses: $(a \vee b \vee \bar{z})$, $(c \vee d \vee z)$, $(\bar{a} \vee \bar{b} \vee z)$, $(\bar{a} \vee b \vee z)$ and $(a \vee \bar{b} \vee z)$. It follows by a standard argument (omitted for brevity) that ψ has the same number of satisfying truth assignments as φ, and every truth assignment of φ has a unique extension to a satisfying truth assignment of ψ. Hence the Horn closure of ψ in any direction has size at least $2^{\lceil m/2 \rceil}$. Thus ψ has $n = \Theta(m^4)$ variables, $\Theta(m)$ satisfying truth assignments and its Horn closure in every direction has size at least $2^{\lceil m/2 \rceil}$, so the theorem follows. □

It may be of interest to note that the bound of Lemma 1 is fairly tight.

Theorem 2. *For every polynomial p and every $\epsilon > 0$, for all sufficiently large m, for every set S of at most $p(m)$ binary vectors of length m, there exists a direction d such that the size of the d-Horn closure of S is at most $2^{\frac{m}{2}(1+\epsilon)}$.*

Proof. We show that a randomly chosen direction $d \in \{0,1\}^m$ has nonzero probability of having the desired property. For every vector $a \in S$, the probability that $a \oplus d$ has more than $\frac{m}{2}(1+\frac{\epsilon}{2})$ 1's is at most $e^{-\epsilon^2 m/8}$ using a Chernoff bound [13, Additive Form, page 190]. If m is sufficiently large then $p(m)e^{-\epsilon^2 m/8} < 1$. In this case there is a direction d such that $a \oplus d$ has at most $\frac{m}{2}(1+\frac{\epsilon}{2})$ 1's for every $a \in S$. Every vector in the d-Horn closure of S is below one of the vectors $a \oplus d$. Hence the size of the d-closure is at most $p(m)2^{\frac{m}{2}(1+\frac{\epsilon}{2})}$, which is less than $2^{\frac{m}{2}(1+\epsilon)}$ for all sufficiently large m. □

The following result shows the existence of CNF formulas for which the Horn-LUB in every direction d gives an incorrect answer to a large fraction of the prime implicate queries. The construction is based on a construction of Levin [15] of a DNF formula with a bounded number of terms, having the maximal number of prime implicants. In [23], we showed that all bounded term DNF with the maximal number of prime implicants can be obtained as a natural generalization of this example.

Theorem 3. *There are polynomial size CNF formulas φ such that for every direction d, the ratio of the number of non-Horn and Horn prime implicates of $\mathcal{R}_d(\varphi)$ is superpolynomial.*

Proof. To construct φ, we begin with a complete binary tree of the height k and put $n = 2^k$. The variables of φ are $x_1, \ldots, x_{n-1}$ and $y_1, \ldots, y_n$. Each internal node of the tree is labeled with a distinct x variable and the ith leaf is labeled with $\bar{y}_i$. The formula φ has n clauses, one for each leaf. The clause corresponding to a leaf is the disjunction of all the variables on the root-to-leaf path to the leaf, with each x variable being negated if and only if the path went left when leaving that node. Thus the depth-1 tree pictured in Figure 1 corresponds to $(\bar{x}_1 \vee \bar{y}_1) \wedge (x_1 \vee \bar{y}_2)$.

The formula φ has a distinct prime implicate for each of the $2^n - 1$ nonempty subsets of the leaves [15, 23]. The prime implicate corresponding to a particular subset S of leaves is the disjunction of x variables corresponding to any inner node such that *exactly one* of the two subtrees of the node contains a leaf in S, and the negated y variables corresponding to the leaves in S. An x variable in the prime implicate is negated iff its left subtree is the one containing leaves in S. Thus, for example, the formula corresponding to the tree in Figure 1 has three prime implicates, one for each nonempty subset of the two leaves. For $\{\bar{y}_1\}$ we have $(\bar{x}_1 \vee \bar{y}_1)$; for $\{\bar{y}_2\}$ we have $(x_1 \vee \bar{y}_2)$; for $\{\bar{y}_1, \bar{y}_2\}$ we have $(\bar{y}_1 \vee \bar{y}_2)$.

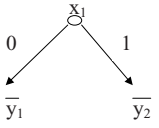


Fig. 1. Tree of depth 1

We give an upper bound for the number of Horn prime implicates under any renaming d. Notice that by symmetry, renaming internal nodes does not change the number of Horn or non-Horn prime implicates. At the leaves, making all the y's negated maximizes the number of the prime implicates that are Horn. Thus it is in fact sufficient to estimate the number of Horn prime implicates of the original formula.

Let H_k (resp., N_k, D_k) be the number of Horn (resp., negative, definite) prime implicates of the formula built from a binary tree of height k. Then $H_k = N_k + D_k$. The numbers H_k satisfy the following recurrence: $H_1 = 3$ and

$$H_k = H_{k-1} + N_{k-1} + H_{k-1} \cdot N_{k-1} + N_{k-1} \cdot D_{k-1} \ . \tag{1}$$

Here the first item is the number of Horn implicates of the left subtree. The second term is the number of negative Horn implicates of the right subtree: by adding the unnegated variable from the root, those correspond to definite Horn implicates. The third (resp, the fourth) term corresponds to prime implicates obtained from an arbitrary Horn (resp., a negative) prime implicate of the left subtree and a negative (resp., a definite) one form the right subtree. Note that two definite prime implicates from the two subtrees will form a non-Horn prime implicate. In order to use (1) to get an upper bound on H_k, we must bound N_k. Similarly to (1), one can derive the following recurrence: $N_1 = 2$ and

$$N_k = (N_{k-1})^2 + N_{k-1}.$$

It can be shown that $N_k < 2^{\frac{11}{16}n}$ and $H_k \leq 3^{k-4} \cdot 2^{14} \cdot 2^{\frac{11}{16}n}$. □

4 Computational Results

Selman and Kautz's original Horn-LUB algorithm for a set of clauses (i.e., a CNF) proceeds by repeatedly performing resolution steps between two clauses,

at least one of which must be non-Horn. Any clauses in the set that are subsumed by the new resolvent are removed, and the new resolvent is added to the set. This process continues until it becomes impossible to find two clauses to resolve such that at least one is non-Horn and their resolvent is not subsumed by some clause already in the set.

In addition to Selman and Kautz's original Horn-LUB algorithm, we considered three other algorithms to compute Horn approximations, which are modifications of the original algorithm:

Renamed-Horn-LUB finds a renaming of the variables using a heuristic algorithm of [3], and then applies the Horn-LUB algorithm. Notice that we need only linear time to find a renaming.

4-Horn-UB works as the Horn-LUB algorithm, but only performs resolution steps that produce clauses of size at most 4.

Renamed-4-Horn-UB is the combination of the first two algorithms: it first performs a renaming, and then does those resolution steps that produce clauses of size at most 4.

It turns out that Renamed-4-Horn-UB gives the best performance, so we give a snapshot of its running time and of the size of its output versus that of the original Horn-LUB in Table 1. (All running times reported in this paper were measured on a Dell laptop with a 2.40 GHz CPU and 256MB RAM.)

Table 1. Mean running time in CPU seconds and number of clauses in the output for Horn-LUB and Renamed-4-Horn-UB on random 3-CNF formulas on 20 variables as a function of density α, averaged over 50 runs

	Original LUB		Renamed-4	
α	Time	Size	Time	Size
1	0.96	96.1	0.00	28.2
2	50.49	1044.7	0.16	236.2
3	126.81	889.8	1.49	704.8
4	224.56	409.3	0.91	452.7

As mentioned in the introduction, we chose to make most of our measurements on formulas on $n = 20$ variables. We restricted n to 20 for two main reasons. First, the running time of the original Horn-LUB algorithm increased by roughly a factor of 5–10 for every additional two variables. So, while by using significantly greater computational resources we could have computed the original Horn-LUB for formulas with 25 variables, it would have been completely infeasible to do so for, say, 75 variables. (Selman and Kautz reported empirical data only on such things as the *unit clauses within the Horn-LUB*, which can be computed much more quickly by using a SAT solver, not by the Horn-LUB algorithm itself.) Second, in several cases we needed to perform exhaustive testing over all 2^n vectors, and this testing becomes impractical for values much above $n = 20$. (More discussion of this issue is given at the end of this section.)

We observe that the running time of Renamed-4-Horn-UB is significantly smaller than for Horn-LUB, and the size of the output formulas is smaller for Renamed-4-Horn-UB for density $\alpha \leq 3$, and modestly larger for density 4. The output sizes for both algorithms are unimodal as a function of CNF density. More detailed data show that the maximum size occurs around density 2.5.

As all these algorithms produce a conjunction of some implicates of the original formula φ, their output is implied by φ; that is, each algorithm's output has a one-sided error. The *relative error* of such an algorithm A on an input formula φ is measured by

$$r_A(\varphi) = \frac{|T(A(\varphi))| - |T(\varphi)|}{|T(\varphi)|},$$

where $A(\varphi)$ denotes the formula output by A on φ.

Figure 2 presents computational results for the relative errors of the four algorithms for different densities on 20 variables. Statistical values on all the figures are median, max and min values; the values at the ends of the white bars are 25% and 75%. The error curves are again unimodal, with maxima around density 2.4. Experiments for fewer variables show similar values of the maxima.

Of the two heuristics taken alone, renaming improves the relative error more dramatically than limiting clauses to size 4; notice that the relative errors were sufficiently different that the two parts of Figure 2 for the renaming heuristic use a different scale. The overall conclusion is that Renamed-4-Horn-UB is the best algorithm for 20 variables.[3] It is significantly faster than either Horn-LUB or Renamed-Horn-LUB, and it is even somewhat faster than 4-Horn-UB. Its output size is significantly smaller than those of Horn-LUB or Renamed-Horn-LUB, but larger than that of 4-Horn-UB. On the other hand, its relative error is only slightly worse than that of Renamed-Horn-LUB, which has the smallest relative error. Replacing the limit 4 on clause size with 3, or even using all implicates of size at most 3, results in a large increase in the relative error for densities below the satisfiability threshold.

It is to be expected, and it is supported by some experimental evidence, that as the number of variables increases, the limit on the clause size required for producing reasonable relative error will also increase.

Another way to evaluate the algorithms is to consider the number of queries that are answered incorrectly by their output. Notice that by Proposition 1, if we have used renaming, we can simply query the renamed clause. We will use the prime implicates of the original formula φ as our test set of clauses. The original Horn-LUB algorithm gives the correct answer for any Horn clause query, and the wrong answer for any non-Horn prime implicate query. Thus the renaming heuristic will improve the query-answering accuracy of the LUB. Restricting the length of resolvents in the upper bound, on the other hand, will worsen the accuracy, as some Horn prime implicates may receive the wrong answer. We show the performance of the four algorithms in Figure 3; these are error

[3] Preliminary experiments show Renamed-4-Horn-UB also performing relatively well for up to at least 40 variables, but for 40 variables, simply *measuring* performance is computationally expensive.

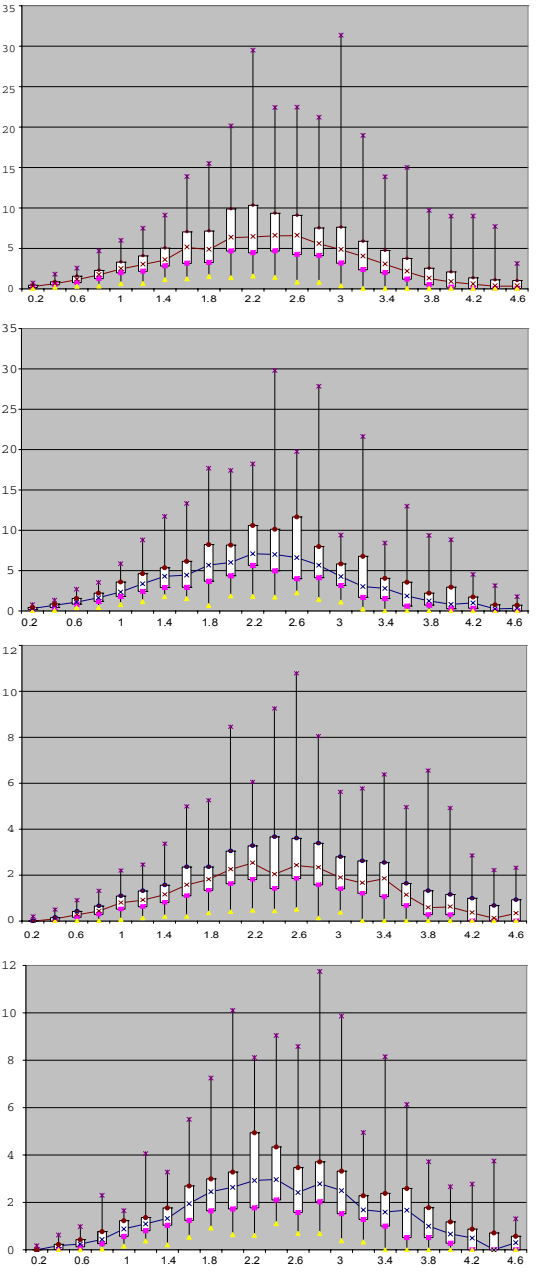


Fig. 2. Relative errors $r_A(\varphi)$ of the algorithms for random 3-CNF with 20 variables as function of density. Measured by exhaustive examination of all length 20 vectors. Averaged over 100 runs. From top to bottom: Horn-LUB, 4-Horn-UB, Renamed-Horn-LUB, Renamed-4-Horn-UB. *The scales for the relative error run from 0–35 for the first two algorithms, but from 0–12 for the Renamed variants.*

ratios, and all are fairly high for densities significantly below the critical density of $\alpha \approx 4.2$. We again observe unimodal behavior, with the maximum around a density of 1.6, with some variation by algorithm. The best performance is indeed for Renamed-Horn-LUB, but Renamed-4-Horn-UB is only a little worse than Renamed-Horn-LUB.

For a larger number of variables it is not feasible to exhaustively measure the behavior of a Horn approximation on all truth assignments. In order to estimate the relative error, one could try to use random sampling by generating a random satisfying truth assignment of the Horn upper bounds. This raises the question whether a random satisfying truth assignment of a Horn formula can be generated (almost) uniformly in polynomial time. As far as we know, this is open. In related work, Roth [20] showed that it is NP-hard to approximate the number of satisfying truth assignments of a Horn formula within a multiplicative factor of $2^{n^{1-\epsilon}}$ (for any ϵ) in polynomial time, even if the clauses have size 2 and every variable occurs at most 3 times, and Jerrum et al. [11] established a connection between almost uniform generation and randomized approximate counting.

Table 2. Percentage error in measuring $r_A(\varphi)$ using "pseudo-random" sampling of vectors versus exhaustive. Generates 100,000 "pseudo-random" length 20 vectors; stops early if 50,000 *distinct* vectors obtained; uses only distinct vectors to measure error. Averaged over 10 runs.

α	0.2	0.6	1	1.4	1.8	2.2	2.6	3	3.4	3.8	4.2
%	1.3	5.4	12	12.9	11.8	8.4	4.2	2.3	1.3	0.0	0.0

Table 3. Percentage error in measuring $r_A(\varphi)$ using "pseudo-random" sampling of vectors using weighted selection of variables versus exhaustive. Generates 100,000 "pseudo-random" length 20 vectors; stops early if 50,000 *distinct* vectors obtained; uses only distinct vectors to measure error. Averaged over 10 runs.

α	0.25	0.65	1.05	1.45	1.85	2.25	2.65	3.05	3.45	3.85	4.25
%	3.3	9.1	11.57	11.5	13.4	9.5	3.3	0.4	0.1	0.0	0.0

We have started to do some initial experiments with various heuristics for generating a satisfying truth assignment of a Horn formula. Table 2 compares the relative error of the Horn-LUB algorithm with the estimate of the relative error obtained by "pseudo-random" sampling. The algorithm is a naive one, randomly selecting variables to be fixed (assigning the same probability to each variable), and deriving all assignments that are forced by the previous choices.

The second algorithm is similar to the first but using a different probability distribution over the variables: the probability assigned to each variable is proportional to the number of its occurrences in the formula. Table 3 also compares the relative error of the Horn-LUB algorithm with the estimate of the relative error obtained by the second algorithm.

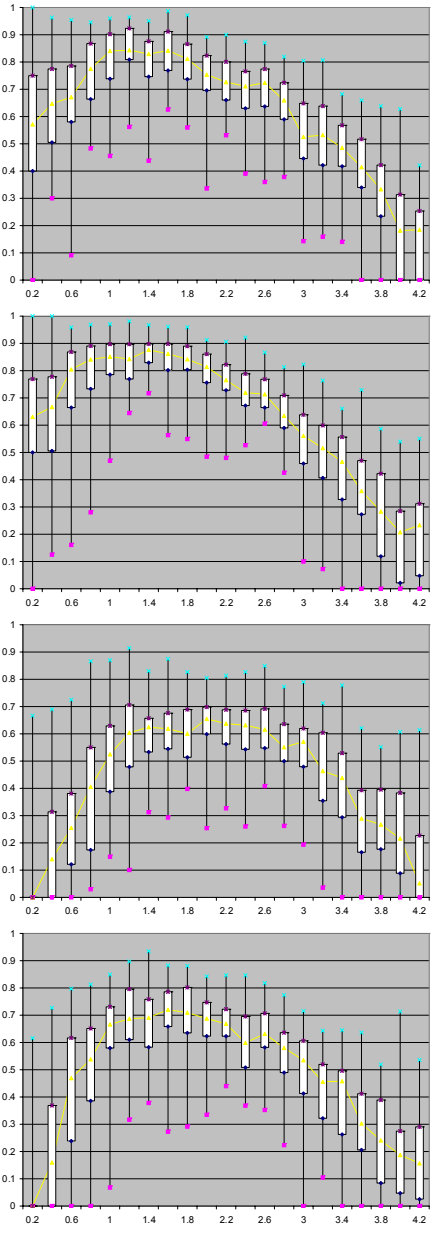


Fig. 3. Fraction of all prime implicate queries to a random 3-CNF formula on 20 variables that would receive the wrong answer from the particular type of Horn upper bound, as a function of density. Averaged over 50 runs. From top to bottom: Horn-LUB, 4-Horn-UB, Renamed-Horn-LUB, Renamed-4-Horn-UB.

Note that truly uniform random generation would require weighting the choices of the two values by the number of satisfying truth assignments corresponding to each value. As Tables 2 and 3 show, the error estimates obtained by "pseudo-random" sampling are rather close to the actual values.

5 Further Remarks

We have given negative results on the approximation quality of Horn upper bounds using renaming, and we have presented experimental results for algorithms generating Horn upper bounds. Based on experiments with random 3-CNF for different densities, we have concluded that for 20 variables the algorithm Renamed-4-Horn-UB provides the best compromise in terms of running time, output size and relative error. Also, a Horn bump was observed for the different performance measures in an intermediate range of densities.

There are several directions for further work. An interesting theoretical problem is to construct CNF expressions having superpolynomially large Horn-LUB for every direction (an example for the ordinary Horn closure is given in [22]). The question of almost uniform random generation of a satisfying truth assignment of a Horn formula seems to be of interest from the point of view of extending the experiments to more variables, and also as a question in itself.

The interpretation of the Horn least upper bound as the intersection closure of the set of satisfying truth assignments for a Horn formula leads to the following general question: what is the expected size of the intersection closure of a random subset of $\{0,1\}^n$, given a probability distribution on the subsets? We are not aware of results of this kind. (Another closure problem, the dimension of the subspace spanned by a random set of vectors, has been studied in great detail.) A basic case to consider would be when m random vectors are generated, each component of which is set to 1 with probability p. The size of the Horn-LUB of a random 3-CNF with a given density is a special case of the general question, when the distribution is generated by picking a random formula.

A much studied problem related to the phase transition of random 3-CNF is the evolution of random 3-CNF, in analogy to the classic work of Erdős and Rényi [9] on the evolution of random graphs, and to the study of the evolution of random Boolean functions (see, e.g., Bollobás et al. [2]). In this direction, all of Mora et al. [19], Mézard and Zecchina [18], and San Miguel Aguirre and Vardi [21] show some interesting behavior at densities below the critical density (clustering of the solutions in the case of Mora et al., a particular behavior of a class of local search algorithms in the case of Mézard and Zecchina, and running times for various solvers in the case of San Miguel Aguirre and Vardi). It would be interesting to know if the Horn bump has any connections to these phenomena.

Acknowledgment. We would like to thank Eli Ben-Sasson and Bart Selman for useful comments.

References

1. B. Aspvall. Recognizing disguised NR(1) instances of the satisfiability problem. *Journal of Algorithms*, 1:97–103, 1980.
2. B. Bollobás, Y. Kohayakawa, and T. Łuczak. On the evolution of random Boolean functions. In P. Frankl, Z. Füredi, G. Katona, and D. Miklós, editors, *Extremal Problems for Finite Sets* (Visegrád), volume 3 of *Bolyai Society Mathematical Studies*, pages 137–156, Budapest, 1994. János Bolyai Mathematical Society.
3. E. Boros. Maximum renamable Horn sub-CNFs. *Discrete Appl. Math.*, 96-97: 29–40, 1999.
4. Y. Boufkhad. Algorithms for propositional KB approximation. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 280–285, 1998.
5. M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3–4):137–150, 1997.
6. Y. Crama, O. Ekin, and P. L. Hammer. Variable and term removal from Boolean formulae. *Discrete Applied Mathematics*, 75(3):217–230, 1997.
7. A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.
8. A. del Val. First order LUB approximations: characterization and algorithms. *Artificial Intelligence*, 162(1-2):7–48, 2005.
9. P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.
10. A. Horn. On sentences which are true on direct unions of algebras. *J. Symbolic Logic*, 16:14–21, 1951.
11. M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.
12. H. Kautz and B. Selman. An empirical evaluation of knowledge compilation by theory approximation. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 155–161, 1994.
13. M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Massachusetts, 1994.
14. M. Langlois, R. H. Sloan, and G. Turán. Horn upper bounds of random 3-CNF: A computational study. In *Ninth Int. Symp. Artificial Intelligence and Mathematics*, 2006. Available on-line from URL `http://anytime.cs.umass.edu/aimath06/`.
15. A. A. Levin. Comparative complexity of disjunctive normal forms. *Metody Discret. Analiz.*, 36:23–38, 1981. In Russian.
16. H. R. Lewis. Renaming a set of clauses as a Horn set. *J. ACM*, 25:134–135, 1978.
17. J. C. C. McKinsey. The decision problem for some classes without quantifiers. *J. Symbolic Logic*, 8:61–76, 1943.
18. M. Mézard and R. Zecchina. The random K-satisfiability problem: from an analytic solution to an efficient algorithm. *Physical Review E*, 66:056126, 2002.
19. T. Mora, M. Mézard, and R. Zecchina. Pairs of SAT assignments and clustering in random Boolean formulae, 2005. Submitted to *Theoretical Computer Science*, available from URL `http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cond-mat/0506053`.
20. D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82: 273–302, 1996.

21. A. San Miguel Aguirre and M. Y. Vardi. Random 3-SAT and BDDs: The plot thickens further. In *Proc. Seventh Int. Conf. Principles and Practice of Constraint Programming (CP)*, pages 121–136, 2001.
22. B. Selman and H. Kautz. Knowledge compilation and theory approximation. *J. ACM*, 43:193–224, 1996.
23. R. H. Sloan, B. Szörényi, and G. Turán. On k-term DNF with the maximal number of prime implicants. Submitted for publication. Preliminary version available as Electronic Colloquium on Computational Complexity (ECCC) Technical Report TR05-023, available on-line at `http://www.eccc.uni-trier.de/eccc/`.
24. K. Truemper. *Effective Logic Computation*. Wiley-Interscience, 1998.
25. H. van Maaren and L. van Norden. Hidden threshold phenomena for fixed-density SAT-formulae. In *Proc. Int. Conf. Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2219 of *Springer LNCS*, pages 135–149, 2004.

Matched Formulas and Backdoor Sets*

Stefan Szeider

Department of Computer Science, Durham University,
Durham DH1 3LE, England, United Kingdom
stefan.szeider@durham.ac.uk

Abstract. We study parameterizations of the satisfiability problem for propositional formulas in conjunctive normal form. In particular, we consider two parameters that generalize the notion of matched formulas: (i) the well studied parameter maximum deficiency, and (ii) the size of smallest backdoor sets with respect to certain base classes of bounded maximum deficiency. The simplest base class considered is the class of matched formulas. Our main technical contribution is a hardness result for the detection of weak, strong, and deletion backdoor sets. This result implies, subject to a complexity theoretic assumption, that small backdoor sets with respect to the base classes under consideration cannot be found significantly faster than by exhaustive search.

1 Introduction and Background

A CNF formula is *matched* if one can match each clause to a "private" variable that occurs in the clause such that different clauses are matched to different variables. Matched CNF formulas are satisfiable since one can satisfy each clause independently by choosing the right truth value for its private variable. Moreover, such formulas can be recognized efficiently by bipartite matching algorithms. Matched formulas play a prominent role in several theoretical investigations. For example, they were used in Tovey's classical paper on 3SAT with bounded occurrence of variables [15], and in Tarsi's Lemma on the clause-variable difference of minimal unsatisfiable formulas [1].

The notion of *maximum deficiency* (first used by Franco and Van Gelder [7] in the context of CNF formulas) allows to gradually extend the nice properties from matched CNF formulas to more general classes of formulas. The maximum deficiency of a CNF formula F, denoted by $\mathrm{md}(F)$, is the smallest number of clauses remaining without a private variable in an optimal matching. The term "maximum deficiency" is motivated by the equality $\mathrm{md}(F) = \max_{F' \subseteq F} \mathrm{d}(F')$ which follows from Hall's Theorem; here $\mathrm{d}(F')$ denotes the *deficiency* of F', the difference between the number of clauses and the number of variables of F'. Let us denote the class of CNF formulas with maximum deficiency at most r by $\mathcal{M}_r$ (thus $\mathcal{M}_0$ is the class of matched formulas).

For minimal unsatisfiable formulas (that is, unsatisfiable formulas that become satisfiable by removing any clause), deficiency and maximum deficiency agree. Kleine Büning [9] initiated the study of minimal unsatisfiable formulas parameterized by deficiency. Fleischner, Kullmann, and Szeider [5] showed that for every constant r, one

* Research supported by the EPSRC, project EP/E001394/1.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 94–99, 2007.

can decide the satisfiability of CNF formulas with maximum deficiency r in polynomial time (as a consequence, minimal unsatisfiable formulas with deficiency bounded by a constant can be recognized in polynomial time). The order of the polynomial that bounds the running time of Fleischner et al.'s algorithm depends on r, which makes the algorithm infeasible for larger inputs, even if r is small. Szeider [13] developed an algorithm that overcomes this limit: it decides satisfiability of CNF formulas with n variables and maximum deficiency r in time $O(2^r n^3)$ and recognizes minimal unsatisfiable formulas with deficiency r in time $O(2^r n^4)$. Thus, satisfiability is *fixed-parameter tractable* with respect to the parameter maximum deficiency, since the degree of the polynomial is independent of the parameter r. We provide some basic concepts of parameterized complexity in Appendix B; for more background information we refer the reader to other sources [4,6,10].

Backdoor Sets

The notion of backdoor sets, introduced by Williams, Gomes, and Selman [16], allows a gradual generalization of any tractable base class $\mathcal{C}$ of CNF formulas. In this paper we consider backdoor sets with respect to the base classes $\mathcal{M}_r$ as defined above. The size of such backdoor sets together with the base class level r provide a two-layered parameterization of the satisfiability problem. A somewhat similar two-layered parameterization of Bayesian reasoning problems was studied by Bidyuk and Dechter [2].

Our results indicate that backdoor sets provide a more general parameterization than maximum deficiency, but one has to pay for more generality with a significantly higher computational cost.

Let us briefly give the definition of backdoor sets (our notational conventions for formulas and truth assignments are given in Appendix A). Let F be a CNF formula and B a set of variables of F. If for every truth assignment $\tau : B \to \{0,1\}$, the restriction $F[\tau]$ belongs to $\mathcal{C}$, then B is called a *strong backdoor set* with respect to the tractable class $\mathcal{C}$ under consideration (or *strong $\mathcal{C}$-backdoor set*, for short). If for at least one $\tau : B \to \{0,1\}$ the formula $F[\tau]$ is satisfiable and belongs to $\mathcal{C}$, then B is called a *weak backdoor set*.

A variant of strong backdoor sets are *deletion* backdoor sets: B is a deletion backdoor set if the formula $F - B$ belongs to $\mathcal{C}$; $F - B$ denotes the formula obtained from F by removing all literals $x, \neg x$ with $x \in B$ from the clauses of F. If the base class is *clause-induced* (i.e., if $F \in \mathcal{C}$ implies $F' \in \mathcal{C}$ for all $F' \subseteq F$), then every deletion backdoor set is a strong backdoor set [12].

If we know a strong or weak backdoor set B of F (or a deletion backdoor set B if the base class is clause-induced), then we can decide satisfiability of F by checking the satisfiability of at most $2^{|B|}$ formulas $F[\tau]$ that belong to the tractable base class. Thus it is interesting to find for a given formula a small backdoor set, say of size at most k. Of course we can consider all sets of variables up to size k, and check whether it is indeed a backdoor set. This exhaustive search requires time of order n^k, thus it becomes infeasible for large n. Whether we can do significantly better than exhaustive search can be studied within the framework of parameterized complexity. To this aim, we formulate the following parameterized decision problems.

WEAK/STRONG/DELETION $\mathcal{C}$-BACKDOOR SET
Instance: A CNF formula F and a non-negative integer k.
Parameter: k.
Question: Does F have a weak/strong/deletion $\mathcal{C}$-backdoor set of size at most k?

Nishimura, Ragde, and Szeider [11] show that for $\mathcal{C} = \text{HORN}$ and $\mathcal{C} = \text{2CNF}$ strong and deletion backdoor sets actually are the same. Furthermore, they show that WEAK $\mathcal{C}$-BACKDOOR SET is W[2] hard and that STRONG $\mathcal{C}$-BACKDOOR SET is fixed-parameter tractable for these two classes.

2 Results

First we consider $\mathcal{M}_0$, the class of matched formulas, as the base class. Since all formulas in $\mathcal{M}_0$ are satisfiable, it follows that every strong $\mathcal{M}_0$-backdoor set is also a weak $\mathcal{M}_0$-backdoor set. Our first result shows that the size of smallest weak $\mathcal{M}_0$-backdoor sets is a parameter that properly generalizes the maximum deficiency parameter.

Theorem 1. *Every satisfiable CNF formula with maximum deficiency k has a weak $\mathcal{M}_0$-backdoor set of size at most k, whereas the difference between the maximum deficiency and the size of a smallest weak $\mathcal{M}_0$-backdoor set can be arbitrarily large.*

Proof. The first part of the theorem follows immediately from a nontrivial result of Fleischner et al. [5] regarding "matching assignments," see also Theorem 1 of [13].

For the second part of Theorem 1 we construct for every positive integer n a CNF formula F such that $\text{md}(F) - |B| > n$ for a smallest $\mathcal{M}_0$-backdoor set B of F. We take variables $x, y_1, \ldots, y_{n+3}$ and consider the formula F consisting of the clauses $\{x, y_i\}$, $\{x, \overline{y_i}\}$, for $i = 1, \ldots, n+3$. The maximum deficiency of F is $2(n+3) - (n+3) - 1 = n + 2$, however $B = \{x\}$ is evidently a weak $\mathcal{M}_0$-backdoor set of F. □

Now we turn our attention to classes $\mathcal{M}_r$ for $r \geq 0$. Note that already $\mathcal{M}_1$ contains unsatisfiable formulas (e.g., $\{\emptyset\}$ or $\{\{x\}, \{\neg x\}\}$), thus not every strong $\mathcal{M}_r$-backdoor set is a weak $\mathcal{M}_r$-backdoor set.

If r and k are fixed constants, then we can detect strong/weak/deletion $\mathcal{M}_r$-backdoor sets of size at most k in polynomial time, since we can search through all sets of variables of size at most k and check the respective conditions. Thus, in contrast to other approaches [14], the backdoor set approach allows a (non-uniformly) tractable generalization of matched CNF formulas. As our main technical contribution, we show that we cannot improve significantly upon exhaustive search, subject to the complexity theoretic assumption FPT $\neq$ W[2]. There are several reasons to believe that the latter assumption holds. For example FPT $=$ W[2] would imply that the Exponential Time Hypothesis fails [6] (i.e., the existence of a $2^{o(n)}$ algorithm for n-variable 3SAT). Further reasons that support the assumption are discussed in a recent work of Dantchev, Martin, and Szeider [3].

Theorem 2. *Let r be a fixed non-negative integer. The problems* WEAK, STRONG, and DELETION $\mathcal{M}_r$-BACKDOOR SET are W[2]-hard.

Proof. We give a parameterized reduction from the following W[2]-complete problem (cf. [4]).

HITTING SET
Instance: A collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, a non-negative integer k.
Parameter: k.
Question: Is there a set $H \subseteq \bigcup_{i=1}^{m} S_i$ of size at most k that hits (i.e., intersects with) each S_i, $1 \leq i \leq m$?

Let $(\mathcal{S}, k)$ be an instances of HITTING SET with $\mathcal{S} = \{S_1, \ldots, S_m\}$, $S_i = \{v_i^1, \ldots, v_i^{q_i}\}$ with $|S_i| = q_i$ for $1 \leq i \leq m$, and $V = \bigcup_{i=1}^{m} S_i$.

We are going to construct a CNF formula F such that $\mathcal{S}$ has a hitting set of size k if and only if F has strong/weak/deletion $\mathcal{M}_r$-backdoor set of size at most k. Our general strategy is to construct for each set S_i a formula F_i' with $\mathrm{md}(F_i') = r + 1$, where $S_i \subseteq \mathrm{var}(F_i')$, and to consider the union F of all the formulas F_i'. Every backdoor set of F must involve a variable of F_i' since $\mathrm{md}(F_i') = r + 1$. On the other hand, F_i' will be constructed in such a way, that deleting any variable of $S_i \subseteq \mathrm{var}(F_i')$ reduces the maximum deficiency of F_i' to 0.

For our construction we use three types of variables: *starting variables*, *hitting variables*, and *matching variables*. As hitting variables we use the elements of V, for the other two types we use new variables. For each $i \in \{1, \ldots, m\}$ we proceed as follows. Let s denote the smallest integer such that $2^s - s > r$. We take new starting variables $y_i^1, \ldots, y_i^s$ and form the set F_i of all 2^s different clauses over these variables. Next we define recursively CNF formulas F_i^j, $j = 0, \ldots, q_i$ by setting

$$F_i^0 := F_i, \text{ and}$$
$$F_i^j := \{\, C \cup \{v_i^j\} : C \in F_i^{j-1} \,\} \cup \{\, C \cup \{\neg v_i^1, \ldots, \neg v_i^j\} : C \in F^i \,\} \text{ for } j > 0.$$

By construction $\mathrm{md}(F_i) \geq \mathrm{d}(F_i) > r$, hence also $\mathrm{md}(F_i^{q_i}) > r$ as can be easily verified. We take a set $M_i = \{z_i^1, \ldots, z_i^{d_i}\}$ of $d_i = \mathrm{md}(F_i^{q_i}) - r - 1$ new matching variables and put

$$F_i' = \{\, C \cup M_i : C \in F_i^{q_i} \,\}.$$

The maximum deficiency of F_i' is by construction exactly $r + 1$ since every matching variable of F_i' can be matched to any clause in F_i'. Finally, we obtain the CNF formula $F = \bigcup_{i=1}^{m} F_i'$. Observe that for $i \neq j$, F_i and F_j do possibly share hitting variables, but do not share any starting variables or matching variables. Note also that since r is a constant, the construction of F can be carried out in polynomial time.

We show that the following statements are equivalent:

1. $\mathcal{S}$ has a hitting set of size at most k.
2. F has a weak $\mathcal{M}_r$-backdoor set of size at most k.
3. F has a strong $\mathcal{M}_r$-backdoor set of size at most k.
4. F has a deletion $\mathcal{M}_r$-backdoor set of size at most k.

Suppose that $H \subseteq V$ is a hitting set of $\mathcal{S}$. We claim that H is a deletion, strong, and weak $\mathcal{M}_r$-backdoor set of F. Actually, it suffices to show that H is deletion $\mathcal{M}_r$-backdoor set: since $\mathcal{M}_r$ is clause-induced, every deletion backdoor set is also a strong one,

and since F is satisfiable (say, by setting all matching variables true) every strong backdoor set is also a weak one.

For every $i \in \{1, \ldots, m\}$, H contains a hitting variable v of F'_i. If we delete v, the number of clauses in F'_i gets reduced by 2^s, but the number of variables gets reduced by 1 only. Since $2^s > r$ by assumption and since $\mathrm{md}(F'_i) = r + 1$, it follows $\mathrm{md}(F'_i - \{v\}) = 0$ and therefore $\mathrm{md}(F'_i - H) = 0$. Hence we can find for each of the remaining clauses in $F'_i - H$ a private variable, and we can choose private variables from the set M_i of matching variables only. Thus $F'_i - H$ is a matched formula. It follows now that also $F - H$ is a matched formula since by construction, F'_i and F'_j $(i \neq j)$ use different matching variables. Whence $F - H \in \mathcal{M}_0 \subseteq \mathcal{M}_r$, and so H is indeed a deletion $\mathcal{M}_r$-backdoor set of F. Thus we have shown that statement 1 implies statements 2, 3, and 4.

Now let B be a set of variables of F and let $\tau : B \to \{0, 1\}$ be a truth assignment. Let $F^* \in \{F[\tau], F - B\}$, and assume that $F^* \in \mathcal{M}_r$. We show that there exists a hitting set H of $\mathcal{S}$ with $|H| \leq |B|$. Suppose that there is some $i \in \{1, \ldots, m\}$ such that no variable of F'_i belongs to B. Consequently, F_i must be a subset of F^*. However, since $\mathrm{md}(F'_i) = r + 1$, $\mathrm{md}(F^*) > r$ follows, a contradiction to $F^* \in \mathcal{M}_r$. Hence for every $i \in \{1, \ldots, m\}$, some variable x_i of F'_i belongs to B (possibly $x_i = x_j$ for $i \neq j$). We define a set of variables $H = \{y_1, \ldots, y_n\}$ as follows. If $x_i \in V$, we put $y_i = x_i$, otherwise we pick $y_i \in S_i$ arbitrarily. It follows now that H is a hitting set of $\mathcal{S}$, and $|H| \leq |B|$ by construction. Whence each of the statements 2, 3, and 4 implies statement 1. This concludes the proof of Theorem 2. □

The reduction in the proof above is actually a polynomial-time many-to-one reduction and does not use the full power available for parameterized reductions. Since the non-parameterized version of HITTING SET (where k is part of the input and is not considered as a parameter) is NP-complete [8], it follows that the non-parameterized versions of the problems mentioned in Theorem 2 are NP-hard.

Appendix A. Notation. We consider propositional formulas in conjunctive normal form, *CNF formulas*, represented as sets of clauses. Each clause is a set of *literals*, each literal is a variable or a negated variable. A (partial) *truth assignment* is a mapping $\tau : X \to \{0, 1\}$ defined for some set X of variables. It extends to literals by putting $\tau(\neg x) = 1 - \tau(x)$. The *restriction* of F to τ is the CNF formula $F[\tau]$ that is obtained from F by removing all clauses that contain a literal ℓ with $\tau(\ell) = 1$ and by removing all literals ℓ with $\tau(\ell) = 0$ from the remaining clauses. F is *satisfiable* if $F[\tau] = \emptyset$ for some truth assignment τ. For a set X of variables we write $\overline{X} = \{\, \neg x : x \in X \,\}$. By *deleting* X from a formula F we obtain the formula $F - X = \{\, C \setminus (X \cup \overline{X}) : C \in F \,\}$.

Appendix B. Parameterized Complexity. An instance of a parameterized problem is a pair (I, k) where I is the *main part* and k is the *parameter*; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* if it can be solved by a fixed-parameter algorithm, i.e., if instances (I, k) can be solved in time $O(f(k)\|I\|^c)$ where f is a computable function, c is a constant, and $\|I\|$ represents the size of I in a reasonable encoding. FPT denotes the class of all fixed-parameter tractable decision problems.

A parameterized reduction is a straightforward extension of a polynomial-time many-one reduction that ensures a parameter for one problem maps into a parameter for another. More specifically, problem L reduces to L' if there is a mapping R from instances of L to instances of L' such that (i) (I, k) is a yes-instance of L if and only if $(I', k') = R(I, k)$ is a yes-instance of L', (ii) $k' = g(k)$ for a computable function g, and (iii) R can be computed by a fixed-parameter algorithm, that is, in time $O(f(k)\|I\|^c)$ where f is a computable function and c is a constant. The reduction defined in the proof of Theorem 2 is actually computable in polynomial time and $g(k)$ is the identity mapping.

References

1. R. Aharoni and N. Linial. Minimal non-two-colorable hypergraphs and minimal unsatisfiable formulas. *J. Combin. Theory Ser. A*, 43:196–204, 1986.
2. B. Bidyuk and R. Dechter. On finding minimal w-cutset problem. *UAI 2004*, 20th Conference on Uncertainty in Artificial Intelligence, 2004.
3. S. Dantchev, B. Martin, and S. Szeider. Parameterized proof complexity: a complexity gap for parameterized tree-like resolution. Technical Report TR07-001, *Electronic Colloquium on Computational Complexity* (ECCC), Jan. 2007.
4. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer Verlag, 1999.
5. H. Fleischner, O. Kullmann, and S. Szeider. Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference. *Theoret. Comput. Sci.*, 289(1):503–516, 2002.
6. J. Flum and M. Grohe. *Parameterized Complexity Theory*, Springer Verlag, 2006.
7. J. Franco and A. Van Gelder. A perspective on certain polynomial time solvable classes of satisfiability. *Discr. Appl. Math.*, 125:177–214, 2003.
8. M. R. Garey and D. R. Johnson. *Computers and Intractability*. Freeman & Co., 1979.
9. H. Kleine Büning. On subclasses of minimal unsatisfiable formulas. *Discr. Appl. Math.*, 107(1–3):83–98, 2000.
10. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
11. N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *SAT 2004*, Informal Proceedings, pages 96–103, 2004.
12. N. Nishimura, P. Ragde, and S. Szeider. Solving #SAT using vertex covers. In *Proc. SAT 2006*, volume 4121 of *LNCS*, pages 396–409, 2006.
13. S. Szeider. Minimal unsatisfiable formulas with bounded clause-variable difference are fixed-parameter tractable. *J. of Computer and System Sciences*, 69(4):656–674, 2004.
14. S. Szeider. Generalizations of matched CNF formulas. *Ann. Math. Artif. Intell.*, 43(1-4): 223–238, 2005.
15. C. A. Tovey. A simplified NP-complete satisfiability problem. *Discr. Appl. Math.*, 8(1): 85–89, 1984.
16. R. Williams, C. Gomes, and B. Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *SAT 2003*, Informal Proceedings, pages 222–230, 2003.

Short XORs for Model Counting: From Theory to Practice

Carla P. Gomes[1,⋆], Joerg Hoffmann[2], Ashish Sabharwal[1,*], and Bart Selman[1,*]

[1] Dept. of Computer Science, Cornell University, Ithaca NY 14853-7501, U.S.A
{gomes,sabhar,selman}@cs.cornell.edu
[2] University of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria
joerg.hoffmann@deri.org

Abstract. A promising approach for model counting was recently introduced, which in theory requires the use of large random XOR or parity constraints to obtain near-exact counts of solutions to Boolean formulas. In practice, however, short XOR constraints are preferred as they allow better constraint propagation in SAT solvers. We narrow this gap between theory and practice by presenting experimental evidence that for structured problem domains, very short XOR constraints can lead to probabilistic variance as low as large XOR constraints, and thus provide the same correctness guarantees. We initiate an understanding of this phenomenon by relating it to structural properties of synthetic instances.

1 Introduction

The dramatic advances in Boolean satisfiability or SAT technology have led to the exploration of new possible applications of SAT solvers. Some of the most exciting such applications go beyond pure satisfiability testing. For example, they involve random sampling from the set of satisfying truth assignments and counting the total number of satisfying assignments. These techniques are particularly promising in the context of applications to probabilistic reasoning. Computationally speaking, counting and sampling are considerably harder than satisfiability testing per se. We recently introduced an approach to counting [1] and sampling [2] that relied on adding XOR or parity constraints [3] (converted to the usual CNF form) to the original problem instance. We showed how one can then use standard state-of-the-art SAT solvers running on the augmented problem instance to compute bounds on the model count of the original problem instance, and to sample near-uniformly from the solution space.

This XOR framework provides probabilistic correctness guarantees for XOR constraints of any length. However, the best results are obtained by using "full-length" XORs, i.e., XORs containing half of the variables of the formula. In our experiments with available SAT solvers, we consistently observed that reasoning with long XORs is computationally much more expensive than reasoning

⋆ Supported by IISI, Cornell University, AFOSR grant F49620-01-1-0076.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 100–106, 2007.

with short XORs. This appears to be because long XOR constraints hamper unit propagation on which SAT solvers rely heavily. Therefore, for the practical applicability of our XOR techniques, a key question is whether relatively short XOR constraints can already provide much of the power of full-length XORs, when considering purely the *quality* of model count bounds and solution samples.

Fortunately, this is the case, as we demonstrate in this paper. In particular, we show that while random 1-XORs (single literals) or random 2-XORs (2 variable XORs) may lead to rather weak bounds on solution counts or sample quality due to large variance, the situation improves dramatically when one considers only slightly longer XORs. In fact, good quality bounds and samples can often be obtained with XORs of 5 to 10 variables, even when the original formula has several hundred variables. To demonstrate this, we systematically consider the *variance* — which directly determines the bound quality — of solution counts obtained in repeated runs with XORs of different lengths. We show that the variance decreases drastically beyond length 1 or 2 XORs. We first demonstrate this phenomenon on a range of practical problem instances, and then provide further insights into the trade-offs between solution space structure and the required length of XORs by considering a class of synthetic problem instances.

2 Background

An XOR *constraint* D over a set of Boolean variables V is the logical "xor" or parity of a subset of $V \cup \{1\}$; a truth assignment for V satisfies D iff it sets an *odd number* of elements in D to TRUE. The value 1 allows us to express even parity. For instance, $D = \{a, b, c, 1\}$ is TRUE as an XOR constraint when an even number of a, b, c are TRUE. Our focus will be on formulas which are a logical conjunction of a formula in Conjunctive Normal Form (CNF) and some XOR constraints. The latter are translated into CNF using additional variables.

XOR-based model counting and sampling methods [1,2] work as follows. Given a formula F, one adds (i.e., conjoins) an appropriate number s of randomly chosen XOR constraints to F to create a new formula F'. F', which in expectation can be shown to have a factor 2^s fewer satisfying assignments than F, is then fed to either an off-the-shelf complete SAT solver or to an exact model counter. When large XORs are used, one can use certain probabilistic independence conditions and transform the result into (a bound on) the model count of F or a random solution sample for F, with guarantees. The length of XORs influences the independence assumption and thus affects the quality of the process; formally, with "short" XORs, only a lower bound model count can be guaranteed.

For one of our analytic computations for the "ideal" case of large XORs, we will use random variables which are the sum of indicator random variables: $Y = \sum_\sigma Y_\sigma$, $Y_\sigma \in \{0, 1\}$. Linearity of expectation says that $\mathbb{E}[Y] = \sum_\sigma \mathbb{E}[Y_\sigma]$. When various Y_σ are *pairwise independent*, i.e., knowing Y_{σ_2} tells us nothing about Y_{σ_1}, even variance behaves linearly: $\mathrm{Var}[Y] = \sum_\sigma \mathrm{Var}[Y_\sigma]$.

3 The Setup for Empirical Evaluation

In all our experiments, we vary a parameter $k, 1 \leq k \leq n/2$, and study the *standard deviation* of the quantity $X = 2^s \times residualSolutionCount$ after a certain number s of random XORs of length k are added to a formula F. The expected value of X is known to be the true model count of F. When the probabilistic variance of X is small (specifically, when $\mathrm{Var}[X] \leq \mathbb{E}[X]$), algorithm `MBound` [1] is able to provide much more than lower bounds: it can compute *near-exact* model counts by computing both a lower bound and an upper bound. This condition was formally proved to hold when $k = n/2$. In this empirical study, we define $k \in \{1, 2, \ldots, n/2\}$ to be *(empirically) good* or *sufficient* for F when $\mathrm{Var}_s[X] \leq \mathbb{E}[X]$ even for XORs of length k, where $\mathrm{Var}_s[X]$ denotes the sample standard deviation (**s.s.d.** for short) of X obtained experimentally.

Since formulas vary in the number of variables and the number of solutions, one must normalize for this when comparing the behavior of XORs on different formulas in the same plot. We do so by two means: (1) we plot the s.s.d. of the normalized solution counts, $X' = X/trueCount$, and (2) the number of XORs we use is fewer than $\log_2 trueCount$ by a constant amount. The first condition guarantees consistent expected values, namely, $\mathbb{E}[X'] = 1$ for all formulas. The second condition, as we will see shortly, ensures that as k approaches $n/2$, s.s.d. $[X]$ approaches the same ideal value for all formulas under consideration.

The ideal curve in all our plots corresponds to the standard deviation of the normalized obtained solution count, X', when full-length XORs are used. Note that this is really a single ideal value to which all s.s.d. plots should converge as the length k increases. We plot it as a horizontal line to easily visually infer for what k are XORs of length k already good for the formula under consideration.

We compute the ideal curve analytically as follows. For concreteness, let Y denote the residual model count obtained after adding s random XORs of length $n/2$ to a formula F with 2^{s^*} solutions. In the notation above, $X = 2^s \times Y$, so that $\mathrm{Var}[X] = 2^{2s} \times \mathrm{Var}[Y]$. Following our earlier analysis [1], we can write $Y = \sum_\sigma Y_\sigma$, where the summation is over all solutions σ of F and Y_σ is a 0-1 random variable indicating whether σ is present in the residual solutions. As argued in that analysis, random variables Y_σ are pairwise independent. Also, $\mathbb{E}[Y_\sigma] = 2^{-s}$ and $\mathrm{Var}[Y_\sigma] = 2^{-s}(1-2^{-s}) \approx 2^{-s}$. Because of pairwise-independence, $\mathrm{Var}[Y] = \sum_\sigma \mathrm{Var}[Y_\sigma] = 2^{s^*-s}(1-2^{-s}) \approx 2^{s^*-s}$. It follows that $\mathrm{Var}[X] \approx 2^{s^*+s}$. For the variance of the normalized model count, we get $\mathrm{Var}[X'] = \mathrm{Var}[X]/2^{2s^*} \approx 2^{-(s^*-s)} = 2^{-remainingXors}$, where *remainingXors* is defined as $(s^* - s)$, i.e., the amount by which the number of XORs added was fewer than the number needed to get down to a single solution. The corresponding s.s.d. is s.s.d. $[X'] = \sqrt{2^{-(s^*-s)}}$, ignoring the relatively tiny $(1 - 2^{-s})$ term.

The ideal curve depicting the behavior of XORs of length $n/2$ is therefore shown as the horizontal line s.s.d. $[X'] = \sqrt{1/2^{s^*-s}}$. We note that when $s = s^*$, so that a single solution is expected to survive, s.s.d. $[X']$ becomes 1. The quantity $s^* - s$, which plays an important role in our experiments, will be referred to as the number of *remaining* XORs. As mentioned above, all formulas plotted in a

single figure for comparison will have the same number of remaining XORs, and will therefore converge to the same ideal value as the length of XORs increases.[1]

4 Experimental Results and Discussion

For each formula F on n variables that we consider, we will vary a parameter k within a sub-range of $\{1, 2, \ldots, n/2\}$ on the horizontal axis, and plot on the vertical axis the sample standard deviation of the normalized quantity $X' = (2^s \times \mathit{residualSolutionCount})/\mathit{totalSolns}$ after a certain number s of random XORs of length k are added to F. For each s.s.d. computation (i.e., for each data point in the plots to follow), we used 1,000 samples in most cases to get a reasonable estimate of the true standard deviation for that length, and up to 50,000 samples in some cases for very short XORs. The number of residual solutions was computed using the exact model counter `Relsat` [4].

We present results on formulas from four domains: Latin squares, logistics planning, circuit synthesis, and random formulas (Figs. 1-4). In each case, there is a dramatic drop in the s.s.d. as the length of XORs is increased even slightly.

The **Latin square formulas** considered have 100 to 150 variables each. The most constrained formula, `ls7R30`, has less than 2^5 solutions, while the least constrained one, `ls7R36`, has 2^{14} solutions. These formulas theoretically require XORs of length 50-75 for near-exact model counting with `MBound`. We performed experiments with 3 remaining XORs. Interestingly, we see from Fig. 1 that at lengths 6 to 8, the s.s.d. already drops to the ideal value. The **logistics planning problem** here has 352 variables and roughly 2^{19} solutions. We again see from Fig. 2 that the variance drops sharply till XOR length 25. At lengths 40 to 50, we are already very close to the ideal behavior. The **circuit synthesis formulas** are for finding minimal size circuits for a given Boolean function. We consider the instance `2bitmax_6` with 252 variables and ideal XOR length 126. This formulas has roughly 2^{97} solutions and we used 87 XORs. Fig. 3 shows that while the s.s.d. is fairly high for very short XORs, it drops dramatically as the length increases to 7, and gets very close to the ideal value at around length 10.

Our **random 3-CNF formulas** are selected from the under-constrained region where model counting is known to be computationally hard [4], i.e., with clause-to-variable ratios significantly below the satisfiability threshold of ≈ 4.26. We consider four 100 variable formulas at ratios 3.3, 3.8, 3.96, and 4.2, respectively. The number of solutions ranges from 2^{32} to 2^{14}. The plots in Fig. 4 indicate that random formulas in general show much higher variance than more structured, real-world formulas considered earlier. In particular, the ratio 4.2 formula achieves ideal behavior at XOR length more than 40. On the positive side, as these formulas become less and less constrained, shorter and shorter XORs surprisingly begin to be sufficient. E.g., the ratio 3.3 formula works well

[1] One could alternatively consider plotting various formulas while keeping the number of XORs fixed, rather than keeping the number of remaining XORs fixed. As the above calculation shows, their s.s.d. plots will then eventually converge to different values, making formula-to-formula comparison not very meaningful.

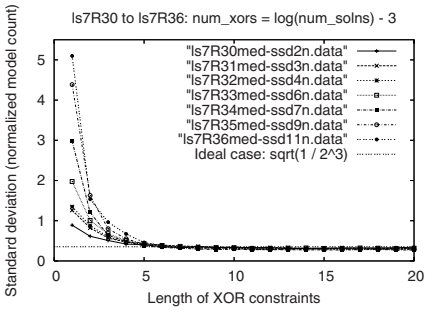


Fig. 1. Latin square formulas of order 7 (100-150 variables)

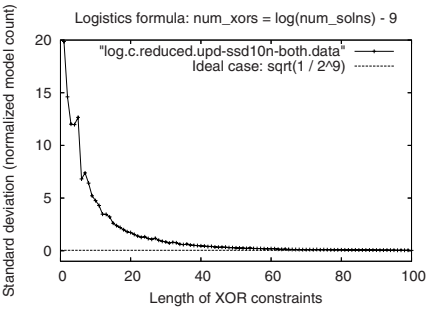


Fig. 2. A logistics planning problem (352 variables after simplification)

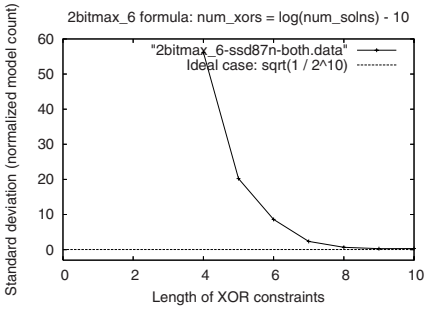


Fig. 3. A circuit synthesis problem (252 variables)

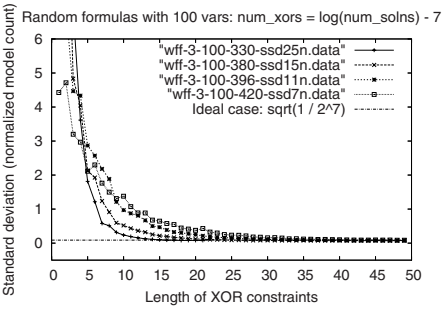


Fig. 4. Random formulas at ratios 3.3, 3.8, 3.96, and 4.2 (100 variables)

even at length 15. This trend suggests that random formulas interestingly become more suitable for shorter XORs as we go into the highly under-constrained region, which is traditionally seen as the harder region for model counting.

In order to better understand the behavior of XOR constraints of various lengths, we explore hand-crafted families of formulas which will help us relate XORs to an intrinsic structural feature of formulas, namely their *backbone*: the set of variables each of which takes the same value in every solution to the formula. We first consider very simple **fixed-backbone formulas** `string-n-t`, with n variables and backbone size $n - t$. The first t variables of the formula are completely unconstrained, while the last $n - t$ variables are fixed to 1. This formula has exactly 2^t solutions, which we will denote by: $1^{n-t} *^t$.

Fig. 5 plots the s.s.d. for these formulas with $n = 50$ variables, $t = 20, 30, 40$, or 49 unconstrained variables, and the corresponding backbone size $30, 20, 10$, or 1. We see that formulas with larger backbone size clearly require larger XORs. This is explained qualitatively by the fact that randomly chosen XORs become more likely to only involve backbone variables as the backbone size increases. When an XOR constraint only involves backbone variables, we encounter unwanted

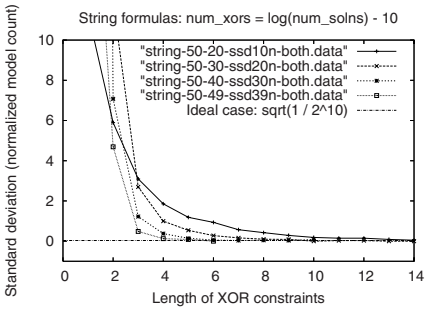


Fig. 5. Fixed-backbones formulas (50 variables)

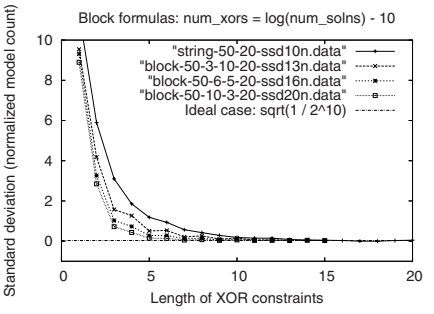


Fig. 6. Interleaved-backbones formulas (50 variables)

behavior: the constraint is either satisfied by all solutions or falsified by all solutions. While this still cuts down the solution space in half on average, there is high variance in the residual count. On the other hand, with small backbones, randomly chosen XORs are very likely to involve at least one non-backbone variable (in this case, one unconstrained variable). When this happens, some of the solutions satisfy the constraint and others don't. This still cuts down the solution space in half on average, but now with lower variance.

The **interleaved-backbones formulas** we consider next attempt to replace a large *global* backbone for all solutions into many overlapping (and conflicting) *local* backbones for solution clusters. These local backbones are interleaved together, allowing all possible combinations of their constituent "blocks," thereby giving the XORs more freedom. These formulas are `block-n-m-k-t`, constructed as follows. There are n variables divided up into m blocks of size k each, and t unconstrained variables ($n = mk + t$). Each block is constrained to have all its variables take the same value. The blocks themselves are, however, independent of each other. We can represent this formula by its solution space: $a_1^k\ a_2^k \ldots a_m^k\ *^t$, where each $a_i \in \{0, 1\}$. Recall that the formula, `string-n-t`, has exactly 2^t solutions. The number of solutions of `block-n-m-k-t` is 2^{t+m}, which increases as the number of blocks in the backbone split is increased, allowing more freedom.

Fig. 6 gives the results for 50 variable block formulas with 30 unconstrained variables and the rest split into 3, 6, and 10 blocks. We see that as the backbone is split into more and more blocks, the variance decreases. In particular, the variance is the highest when there are no blocks (the string formula at the top) and the lowest when the backbone is split into 10 blocks.

5 Concluding Remarks

XOR-streamlining is a promising approach for model counting and sampling. We provided evidence that relatively short XORs can be surprisingly powerful on practical problem instances. While large global backbones are bad for XORs, our synthetic formulas based on interleaved backbones provide intuitive explanation

that solution spaces consisting of many clusters with large local backbones are still fine. We believe that this latter structure is more likely to be present in real-world formulas than large global backbones or uniformly distributed solutions.

References

1. Gomes, C.P., Sabharwal, A., Selman, B.: Model counting: A new strategy for obtaining good bounds. In: 21th AAAI, Boston, MA (2006) 54–61
2. Gomes, C.P., Sabharwal, A., Selman, B.: Near-uniform sampling of combinatorial spaces using XOR constraints. In: 20th NIPS, Vancouver, B.C. (2006)
3. Valiant, L.G., Vazirani, V.V.: NP is as easy as detecting unique solutions. Theoretical Comput. Sci. **47**(3) (1986) 85–93
4. Bayardo Jr., R.J., Pehoushek, J.D.: Counting models using connected components. In: 17th AAAI, Austin, TX (2000) 157–162

Variable Dependency in Local Search: Prevention Is Better Than Cure

Steven Prestwich

Cork Constraint Computation Centre
Department of Computer Science, University College, Cork, Ireland
s.prestwich@cs.ucc.ie

Abstract. Local search achieves good results on a variety of SAT problems and often scales up better than backtrack search. But despite recent advances in local search heuristics it has failed to solve some structured problems, while backtrack search has advanced greatly on such problems. We conjecture that current modelling practices are unintentionally biased in favour of solution by backtrack search. To test this conjecture we remodel two problems whose large instances have long resisted solution by local search: parity learning and Towers of Hanoi as STRIPS planning. By reducing variable dependencies and using other techniques we boost local search performance by several orders of magnitude in both cases, and we can now solve 32-bit and 6-disk instances for the first time using a standard SAT local search algorithm.

1 Introduction

Local search is often more scalable than backtrack search, and in some areas of combinatorial optimisation is the only practical way of obtaining good solutions. Yet it currently has the reputation of being inferior to DPLL (the Davis-Putnam-Logemann-Loveland SAT backtracking algorithm) on structured SAT instances and only good for random problems. Nevertheless, it cannot be denied that local search generally performs badly on problems classed as *industrial* in SAT solver competitions, and the winners are all DPLL variants (see for example [35]). Hybridising local search with unit propagation [11,24] or explicitly handling variable dependencies [15,23] helps, but DPLL is still unbeaten on these problems. Improving local search on structured problems would have many practical applications, perhaps solving larger instances of real-world applications than is currently possible, but we cannot begin to improve it until we understand the cause of its poor performance.

We conjecture that *current SAT-encodings are unintentionally designed to favour DPLL*, and that this explains the poor ranking of local search algorithms in solver competitions. For example, in SAT modelling we often eliminate symmetrical solutions. But when applying local search, symmetry appear to be harmless or even helpful, and eliminating it can adversely affect performance [25,27]. Moreover, SAT encodings of specific constraints have been explored by

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 107–120, 2007.

[1,2,7,8,9,13] with the aim of improving the consistency reasoning of unit propagation in DPLL. But unit propagation is not used in most local search algorithms so consistency reasoning is not necessarily relevant. In fact it was shown in [26] that the *ladder* structure introduced in some of these encodings has a harmful effect on local search performance. We also make a more specific conjecture: that the model feature to blame for local search's poor performance is in many cases *dependent variables*. These are known to slow down local search [15], and when they form long chains they may cause local search to take polynomial or exponential time to propagate effects [24,33].

Our conjectures can be tested empirically by devising new SAT encodings with reduced variable dependency, and comparing local search on the old and new encodings. We do this for two problems whose large instances have so far resisted solution by local search. In Section 2 we reformulate the *parity learning* problem to avoid dependency chains, via new SAT-encodings of parity and cardinality constraints, and show that local search can solve 32-bit instances. In Section 3 we reformulate the *Towers of Hanoi* problem expressed as a planning problem, breaking up long dependency chains into short ones and artificially increasing solution density, and show that local search can solve the 6-disk instance. Both results are firsts for an off-the-shelf SAT local search algorithm.[1] demonstrating the power of the remodelling approach. Section 4 discusses further applications and concludes the paper.

All our experiments are performed on a 733 MHz Pentium II under Linux. We use only one local search algorithm: RSAPS [12] implemented in the UBCSAT system [30]. RSAPS is a state-of-the-art dynamic local search algorithm, and was chosen after preliminary experiments indicated that it was one of the best algorithms for these problem. It also has the advantage that its default parameter settings give good results over a wide range of SAT problems, so we did not need to tune them.

2 Minimal Disagreement Parity Learning

This problem description is taken from [4]. Given vectors $\mathbf{x}_i = (x_{i1}, \ldots, x_{in})$ $(i = 1 \ldots m)$ with each $x_{ij} \in \{0, 1\}$, a vector $\mathbf{y} = (y_1, \ldots, y_m)$ and an error tolerance integer k. Find a vector $\mathbf{a} = (a_1, \ldots, a_n)$ such that $|\{i : \text{parity}(\mathbf{a} \cdot \mathbf{x}_i) \neq y_i\}| \leq k$. To make hard instances set $m = 2n$ and $k = 7n/8$. n is referred to as the number of bits, and 32-bit instances of this problem have proved intractable for both DPLL and local search. They have been solved by paying special attention to the parity constraints, either by transforming many of them away in a preprocessing phase or by augmenting DPLL with equivalence reasoning [2,3,19,32].

Until recently local search has never solved 32-bit instances [16]. A special version of the DLM local search algorithm was created for these and other very hard benchmarks [34] but fails on 32-bit instances, as does a more recent algorithm [22]. It was not until this year that an extended local search algorithm solved them by exploiting knowledge about variable dependencies [23]. Why is

[1] But a recent extended algorithm [23] has solved 32-bit problems — see Section 2.4.

this problem so hard for local search? An explanation suggested by [16] is the existence of local minima in which a small subset of clauses is never satisfied simultaneously. We conjecture that chains of dependent variables are the culprit (this does not necessarily contradict the explanation of [16]) and define a new encoding of parity constraints that contains no such chains.

What we shall call the *standard encoding* is that used in [4] and SATLIB[2] parX-Y-c instances (which are improved versions of the much harder parX-Y instances) and described in [4]. It contains three families of clauses: the first calculates the parities of $\mathbf{a} \cdot \mathbf{x}_i$; the second computes disagreements in parities; the third encodes a cardinality constraint to limit the disagreements (n is set to a power of 2 so that cardinality is easy to enforce). We start with a slightly different model of parity learning that allows experimentation with different encodings of parity constraints. If we can find improved encodings then these may be useful when modelling other problems.

2.1 A Constraint-Based Model

Define variables A_i to contain the solution and P_j to denote parities. Force each scalar product $\mathbf{a} \cdot \mathbf{x}_j$ to have parity P_j:

$$P_j \equiv \bigoplus_{i \in \tau_j} A_i$$

where $\tau_j = \{i \,|\, x_{ij} = T\}$. Then at most k of m literals are true:

$$\mathrm{LE}(k, \pi_1, \ldots, \pi_m)$$

where literal π_j is $\bar{P}_j$ if $y_j = T$ and P_j if $y_j = F$. By using a cardinality constraint we can encode parity learning instances of any size, not just with n a power of 2. We now discuss encodings for the $\bigoplus$ and LE constraints.

2.2 Encoding Parity Constraints

It is possible to SAT-encode a parity constraint $\bigoplus_{i=1}^{p} P_i = k$ simply by enumerating all possible combinations of P_i truth values, together with their parity k. But this creates exponentially many clauses and is only reasonable for small constraints. We shall call it the *exponential encoding*. A more practical method due to [19] decomposes the constraint by introducing new variables:

$$P_1 \oplus z_1 \equiv k \qquad P_2 \oplus z_2 \equiv z_1 \qquad \ldots \qquad P_{p-3} \oplus z_{p-3} \equiv z_{p-2} \qquad P_p \equiv z_{p-1}$$

The remaining binary and ternary constraints are then expanded via the exponential encoding. We shall call this the *linear encoding*. It has $O(p)$ new variables and literals and is essentially the method used to encode the parities of $\mathbf{a} \cdot \mathbf{x}_i$ in the standard encoding of parity learning.

[2] http://www.cs.ubc.ca/~hoos/SATLIB/

A drawback with the linear encoding is that it creates a long chain of variable dependencies, which has been shown to be bad for local search performance [24,33]. An obvious alternative is a divide-and-conquer approach: bisect the constraint, solve the two subproblems, and merge the two results by a ternary constraint. That is, express $\bigoplus_{i=1}^{p} P_i = k$ as $\bigoplus_{i=1}^{p/2} P_i = k_1$, $\bigoplus_{i=p/2+1}^{p} P_i = k_2$ and $k = k_1 \oplus k_2$, and recursively decompose until reaching a base case of size 2 or 3. All binary and ternary parity constraints are expanded into clauses via the exponential encoding. We shall call this the *bisection encoding*. It replaces the chains of dependency of length p by a tree of depth $\log p$.

We also try a third technique. Decompose $\bigoplus_{i=1}^{p} P_i = k$ into

$$\bigoplus_{i=1}^{\alpha} P_i \equiv k_1 \qquad \bigoplus_{i=\alpha+1}^{2\alpha} P_i \equiv k_2 \qquad \ldots \qquad \bigoplus_{i=p-\alpha+1}^{p} P_i \equiv k_\beta \qquad \text{and} \qquad \bigoplus_{i=1}^{\beta} k_i \equiv k$$

where $\beta = \lceil p/\alpha \rceil$ and the tree branching factor α is a number in the range $1 < \alpha < p$. Expand the $\beta + 1$ parity constraints into clauses via the exponential encoding. This creates $O(\beta)$ new variables and $O(\beta 2^\alpha + 2^\beta)$ literals. This still gives a tree of variable dependencies but only of depth 2. We exponentially increase the number of clauses but the number is quite manageable for (say) $p \leq 100$. For larger p it can use a slightly less shallow tree of depth (say) 3 or 4. We shall call this the *shallow encoding* and use trees of depth 2 in our experiments.

2.3 Encoding Cardinality Constraints

We use a new SAT encoding of a cardinality constraints $\mathrm{LE}(k, \pi_1, \ldots, \pi_m)$ that places an upper bound on the number of literals in a given set that are allowed to be true. We use this encoding mainly for convenience (it is very easy to implement).

First consider the special case where the upper bound is 1, so that we have an at-most-one (AMO) constraint. Define new Boolean variables b_k where $k = 1 \ldots \lceil \log_2 m \rceil$. Add clauses

$$\bar{\pi}_i \vee b_k \text{ [or } \bar{b}_k\text{]}$$

if bit k of the binary representation of $i - 1$ is 1 [or 0], where $k = 1 \ldots \lceil \log_2 m \rceil$. This encoding has $O(\log m)$ new variables and $O(m \log m)$ binary clauses. This *bitwise* encoding was defined in [26] and shown to work well with local search; other known encodings either have higher space complexity or interact poorly with local search (because of chains of dependent variables).

Now suppose we want to prevent more than k of literals $\pi_1 \ldots \pi_m$ from being true. Suppose we have k bins. Define $x_{ij} = T$ if π_i is placed in bin j. Every true π_i must be placed in a bin:

$$\pi_i \rightarrow \left(\bigvee_j x_{ij} \right)$$

and no more than one π_i may be placed in a bin:

$$\mathrm{AMO}_i(x_{ij})$$

using the bitwise encoding. Of course this encoding introduces a great deal of symmetry, as the π_i can be permuted among the bins. Here we invoke [25,27]: symmetry does not necessarily harm local search performance, and may even improve it.

This cardinality encoding has already been used in [26] to solve clique problems by SAT local search, but the bin structure was entangled in the clique model and a cardinality encoding was not explicitly described. In future work we will compare it with other known encodings such as that of [2], which create trees of dependent variables that may slow down local search.

2.4 Experiments

We do not have access to the original parity learning instances, only their standard SAT encodings. Instead we generate 30 random instances (using the method described in [4]) of each required size and take the median of 30 runs of RSAPS, one run per instance (except for the expensive 28- and 32-bit instances which use only 10 runs). The aim is to estimate typical local search performance on a typical problem.

par8-X-c

X	flips	secs
1	1,144	0.0014
2	1,518	0.0017
3	3,060	0.0034
4	1,477	0.0018
5	2,339	0.0028

par16-X-c

X	flips	secs
1	13,377,611	20
2	16,533,206	24
3	12,863,749	19
4	8,601,612	13
5	13,505,657	20

par32-X-c

X	flips	secs
1	—	—
2	—	—
3	—	—
4	—	—
5	—	—

Fig. 1. Local search results on SATLIB parity learning instances

Median results for the SATLIB instances are shown in Figure 1 and our encoding results are shown in Figure 2. An entry "—" denotes that more than 1 billion flips are needed while "?" denotes experiments not done. The 8- and 16-bit results for the linear encoding are similar to those for the standard encoding (perhaps slightly better). This is a good sanity check: the linear encoding has similar characteristics to the standard encoding, so any major improvements we obtain will be due to improvements in parity constraint encoding. Extrapolating by applying linear regression to the logarithms of the four flip results, and using a measured flip rate of 373,134 flips per second for 32-bit instances under the linear encoding, we expect RSAPS to take approximately 20 trillion flips and 2 years to solve them: it is unsurprising that no successes have been reported with the standard encoding.

The bisection encoding is hardly better than the linear encoding, which is a surprise: the failure of local search does not seem to be caused purely by the length of the chains of dependency, and trees of dependencies may be almost

flips needed to find a solution

			shallow				
n	linear	bisecting	$\beta = 6$	$\beta = 8$	$\beta = 10$	$\beta = 12$	$\beta = 14$
8	2,517	2,836	1,119	747	904	955	891
12	157,433	83,708	10,089	4,494	4,929	3,905	2,265
16	7,139,810	5,326,518	599,662	59,051	23,630	18,457	25,038
20	223,090,992	156,378,976	11,381,251	3,283,580	460,165	167,936	173,826
24	—	—	282,226,496	35,025,380	16,078,792	3,647,108	?
28	—	—	?	251,928,288	131,614,608	?	?
32	—	—	?	?	1,454,529,796	?	?

seconds needed to find a solution

			shallow				
n	linear	bisecting	$\beta = 6$	$\beta = 8$	$\beta = 10$	$\beta = 12$	$\beta = 14$
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.02	0.12	0.02	0.02	0.03	0.02	0.01
16	11	8.5	1.2	0.22	0.47	0.83	1.03
20	408	297	29	13	11	27	69
24	—	—	788	149	272	693	?
28	—	—	?	2,386	3,640	?	?
32	—	—	?	?	49,633	?	?

Fig. 2. Local search results on new parity learning encodings

as harmful. More research is clearly required into what types of structure are bad for local search performance, but if trees of dependency are harmful then standard cardinality encodings seem to be unsuitable for local search.

The shallow encoding gives much better results and is able to solve 32-bit instances in a few hours. The choice of β has a large effect on performance: roughly speaking, the greater the value of β the fewer flips are required to solve the problems; but higher β also means larger models and thus lower flip rates, so in terms of CPU time there is a trade-off. A reasonable value for these instances is $\beta = 10$. We also experimented on 16-bit instances with two other automated local search algorithms: AdaptNovelty+ was also faster on the new ($\beta = 10$) encoding than on the standard encoding, though less so than RSAPS, while VW behaved similarly on both. In contrast, the DPLL algorithms ZChaff, SATO and SATZ were all slowed down by a factor of at least 100 on the new encoding. Modelling for local search is clearly distinct from modelling for backtrack search.

Though our local search results are vastly improved we have not yet matched the best DPLL performance. But local search performance might be further improved by using similar techniques: directly handling parity constraints during search, or preprocessing the problems to eliminate them. Recently Pham et al. [23] solved the *standard* encodings of the 32-bit instances, using a new non-CNF local search algorithm that exploits problem structure analysis. Their execution times are similar to ours (though on an unspecified machine) and they use an order of magnitude fewer flips than we do on 16-bit instances (but do not provide flip figures for 32-bit instances).

3 Towers of Hanoi as STRIPS Planning

Planning problems expressed in the STRIPS language have been SAT-encoded many times [5,14,17,18]. SAT-based planning has achieved state-of-the-art results on STRIPS [6] planning problems and is one of the success stories of SAT research. We study the Towers of Hanoi (ToH) problem modelled as a STRIPS planning problem. ToH is perhaps not a very interesting problem in itself, and solving it via STRIPS and SAT is certainly not the best approach. Its interest lies in the fact that ToH makes very hard planning problems for local search: harder than the Blocks World instances, which are in turn harder than the logistics instances [28]. In the 2002 SAT solver competition, no local search algorithm solved the ToH problems with 4, 5 and 6 disks, while the BerkMin backtracker solved the 6-disk problem in 2551 sec. A special version of the DLM local search algorithm was created for these and other very hard benchmarks, and solved 4 disks in almost 1 billion flips and over 2 hours [34].

Why is ToH so hard for local search? It may be because it has only one solution [28]. We believe that the explanation is a combination of low solution density and the chain-like structure of dependent variables in SAT-encoded planning problems. We modify both the STRIPs model and the SAT-encoding, increasing solution density and breaking up variable chains, to obtain huge improvements in local search performance. First we define what we shall refer to as a *standard approach* to SAT-encoding ToH as a STRIPS planning problem. There is of course no single standard approach but ours is based on published techniques.

3.1 ToH as STRIPS

The ToH problem consists of P pegs (or towers) and D disks of different sizes; usually $P = 3$. All d disks are initially on the first peg. A solution is a plan that moves all disks to the third peg with the help of the second peg so that (i) only one disk can be moved at a time; (ii) only the disk on top can be moved; (iii) no disk can be put onto a smaller disk. There is always a plan with $2^D - 1$ steps.

We have two fluents $\mathrm{on}(d, dp)$ and $\mathrm{clear}(dp)$, where d denotes a disk and dp either a disk or a peg. We also have an action $\mathrm{move}(d, dp, dp')$ that moves d from dp to dp' with preconditions $\{\mathrm{clear}(d), \mathrm{clear}(dp'), \mathrm{on}(d, dp)\}$, add effects $\{\mathrm{on}(d, dp'), \mathrm{clear}(dp))\}$ and delete effects $\{\mathrm{on}(d, dp), \mathrm{clear}(dp')\}$. In the initial state fluents $\{\mathrm{on}(disk_1, disk_2), \ldots, \mathrm{on}(disk_{D-1}, disk_D), \mathrm{on}(disk_D, peg_1), \mathrm{clear}(disk_1), \mathrm{clear}(peg_2), \ldots, \mathrm{clear}(peg_P)\}$ are true and all others are false. In the goal state fluents $\{\mathrm{on}(disk_1, disk_2), \ldots, \mathrm{on}(disk_{D-1}, disk_D), \mathrm{on}(disk_D, peg_P)\}$ are true and all others have unspecified truth values.

3.2 STRIPS as SAT

We start from an encoding similar to those used in [14,17,18]. First we set an upper bound on the plan length of discrete times $t = 0 \ldots N-1$. Define two sets of variables: τ_{pt} ($t = 0 \ldots N$ where N denotes the state after the last action) and predicate p, where $\tau_{pt} = T$ iff p is true at the start of time t; and α_{at} ($t = 0 \ldots N$)

where $\alpha_{at} = T$ iff action a occurs at time t. The clauses are as follows. Exclusion axioms restrict the plan to be linear, in which at most one action occurs at any time t:

$$\overline{\alpha_{at}} \vee \overline{\alpha_{a't}}$$

We do not force actions to occur at every time, which creates additional solutions (if the plan length is overestimated) that may help local search. Actions imply preconditions:

$$\alpha_{at} \rightarrow \tau_{pt} \quad \text{or} \quad \alpha_{at} \rightarrow \overline{\tau_{pt}}$$

and effects:

$$\alpha_{at} \rightarrow \tau_{p\,t+1} \quad \text{or} \quad \alpha_{at} \rightarrow \overline{\tau_{p\,t+1}}$$

Frame axioms preserve fluents that are unaffected by actions. These may be in either *classical* or *explanatory* form. We use the explanatory form [10] which is more compact [5,14] and also obviates the need for every time slot to contain an action:

$$\tau_{pt} \wedge \overline{\tau_{p\,t+1}} \rightarrow \left(\bigvee_{a \in E_{\bar{p}}} \alpha_{at} \right) \quad \text{or} \quad \overline{\tau_{pt}} \wedge \tau_{p\,t+1} \rightarrow \left(\bigvee_{a \in E_p} \alpha_{at} \right)$$

where E_p ($E_{\bar{p}}$) denotes the set of actions with add (delete) effect p. The initial state is represented by unit clauses:

$$\tau_{p0} \quad \text{or} \quad \overline{\tau_{p0}}$$

as is the goal state:

$$\tau_{pN} \quad \text{or} \quad \overline{\tau_{pN}}$$

We now try to improve the standard approach in several ways.

3.3 Exploiting Domain Knowledge

ToH has been modelled in the same way as a Blocks World problem, but its special form allows a simpler STRIPS model. We retain the *move* operator and *on* predicate as before, but drop the *clear* predicate and only specify which peg a disk is on, not which disk or peg. The action move(d, x, y, t) now has preconditions $\{\text{on}(d, x),\ \neg\text{on}(1, x, t),\ \ldots,\ \neg\text{on}(d-1, x, t),\ \neg\text{on}(1, y, t),\ \ldots,\ \neg\text{on}(d-1, y, t)\}$ for all $d' < d$, add effects $\{\text{on}(d, y, t+1)\}$ and delete effects $\{\text{on}(d, x, t+1)\}$. In the initial state fluents $\{\text{on}(1, 1),\ \ldots,\ \text{on}(D, 1)\}$ are true and all others are false, while in the goal state $\{\text{on}(1, P),\ \ldots,\ \text{on}(D, P)\}$ are true and the others unspecified.

Besides having fewer predicates, this model has fewer actions because each disk can only be on P pegs instead of $D + P$ disks or pegs. In effect we are using the domain knowledge that disks are stacked in decreasing order of size; this trick would not work on general Blocks World problems.

3.4 Superparallelism

An important technique in planning is the use of *parallel plans* in which more than one action may occur at a given time. Besides being more appropriate for some applications, this allows the plan length to be shorter and thus the SAT problem to be smaller. As the size of the SAT problem can be a bottleneck for real-world planning problems, this benefits both DPLL and local search algorithms. Parallelism may have another advantage for local search: it increases the solution density of the SAT problem. This is because any linear plan has multiple representations as a parallel plan, typically an exponential number of them. It has been shown that increasing the solution density of a SAT problem can boost local search performance (though this is not guaranteed).

Unfortunately there is no natural parallelism in ToH. But we can allow some actions to be performed in parallel in the new model, by removing some exclusion axioms:

- Allow (say) disk 1 to move from peg 1 to peg 2, and disk 2 to move from peg 3 to peg 2, at the same time: this can be uniquely transformed to: move the larger disk to peg 2 then the smaller one.
- Allow (say) disk 1 to move from peg 1 to peg 2, and disk 2 to move from peg 2 to peg 3, at the same time. This can be uniquely transformed to: move disk 2 then disk 1. There is no danger of an illegal cycle of three moves as the preconditions will prevent one of the disks from moving onto a peg with a smaller one.

We shall call this *superparallelism* because it adds parallelism beyond any that is naturally present in the model (in this case none). Superparallel moves are illegal and must be transformed away after finding a plan. A drawback with superparallelism is that we can no longer force the search to find optimal plans. Even if we reduce the number of times to the smallest possible value, after transformation we may obtain a very suboptimal linear plan. But it may be a useful technique for applications in which any feasible plan will do, or for quickly obtaining an initial plan for subsequent improvement. Another possibility is to place an upper limit on the total number of actions via a SAT-encoded cardinality constraint.

It is not possible to force parallelism in the standard STRIPS model of Blocks World by dropping exclusion axioms, because performing any two actions at the same time would lead to an inconsistent state. However, it would be possible to define new actions that move more than one disk at a time; we leave this for future work.

3.5 Long-Range Dependencies

The encoding of Section 3.2 has a potential drawback for local search: the frame axioms create chains of dependent variables $\tau_{pt} \ldots \tau_{pt'}$ for each p and pair t, t'. As noted above, dependent variables are known to be a major source of slowdown in local search [15], especially when they occur in chains [24,33].

At first glance there seems to be no way to avoid these chains, as they are a property of the problem itself and not the encoding. However, it is possible to break up the chain structure by using the method of [33]: add implied clauses to cause *long-range dependencies* between times further apart than 1 unit. The clauses we add are a generalisation of explanatory frame axioms to time differences ≥ 1:

$$\tau_{pt} \wedge \overline{\tau_{pt'}} \rightarrow \left[\bigvee_{t''=t}^{t'-1} \left(\bigvee_{a \in E_{\bar{p}}} \alpha_{at''} \right) \right] \quad \text{or} \quad \overline{\tau_{pt}} \wedge \tau_{pt'} \rightarrow \left[\bigvee_{t''=t}^{t'-1} \left(\bigvee_{a \in E_p} \alpha_{at''} \right) \right]$$

where $t' > t$. We shall call these *generalised explanatory frame (GEF) axioms*. The usual explanatory frame axioms are given by the case $t' = t + 1$.

Adding all GEF axioms increases the space complexity, but we can add a randomly-chosen subset of them (but including the usual explanatory frame axioms), by analogy with [33] who found that adding a relatively small number of implied clauses gave optimal improvement.

Unlike the fixed-length added clauses of [33] ours grow with n, so their effect on search time may be inferior. We could reduce their length by defining new variables $\epsilon_{pt} \rightarrow \left(\bigvee_{a \in E_p} \alpha_{at} \right)$ where $\epsilon_{pt} = T$ only if an action with effect p occurs at time t. This allows us to simplify the GEF axioms but in experiments it made the problems harder to solve.

3.6 Implied Clauses

Besides the GEF axioms we add another set of implied clauses: exclusion axioms corresponding to two disks making the same move. This can never occur because the larger disk's preconditions are unsatisfied if the smaller one is on the same peg, so these clauses are redundant.

3.7 Experiments

We compare four models: the standard model, the compact model (using special domain knowledge), the compact model with parallelism, and the compact model with parallelism and GEF axioms. All models use the implied constraints described above. In experiments 5% was approximately the best proportion of randomly-selected GEF axioms, which is less than the 20% figure of [33].[3] All results are medians of 30 runs. In each case the number of times was set to $2^D - 1$, the optimum for a linear ToH plan. The results are shown in Figure 3, with "—" denoting that RSAPS failed to find a solution after 1 billion flips.

The hardness of ToH grows extremely rapidly with D in all models. The compact model gives much better results than the standard model, and parallelism and GEF axioms greatly improve performance. By combining several modelling

[3] Actually, this was for the largest instances, and a higher percentage was better for smaller instances, possibly indicating that the optimum number of GEF axioms is less than linear in the problem size.

local moves (flips)				
D	standard	compact	parallel	GEF
3	38,271	3,730	546	410
4	—	2,757,378	4,866	5,985
5	—	—	532,488	51,453
6	—	—	—	40,163,929

execution time (seconds)				
D	standard	compact	parallel	GEF
3	0.096	0.0058	0.0010	0.0010
4	—	5.8	0.0093	0.017
5	—	—	1.8	0.30
6	—	—	—	980

Fig. 3. Results of experiments

techniques we have obtained the best-reported SAT local search results for 4, 5 and 6 disks, and they are comparable to the best DPLL results (though at the price of reducing plan quality through superparallelism). We also added GEF axioms to the standard model but were still unable to solve 4 disks. We expect further improvements by using the well-known techniques of operator splitting (which reduces the space complexity of SAT-encoded planning problems) and preprocessing by unit propagation and subsumption.

In further local search experiments, AdaptNovelty+ and VW were faster on compact model than on the standard model, and even faster with superparallelism. AdaptNovelty+ was faster with GEF axioms, while VW was hardly affected (apart from the overhead of maintaining the additional clauses). The DPLL algorithms ZChaff, SATZ and SATO were all improved by the compact encoding, ZChaff was faster with superparallelism while SATZ and SATO were slower, and SATO was faster with GEF axioms while ZChaff and SATZ were slower. Thus the compact encoding helps all the algorithms, while the other techniques mostly help local search but have an erratic effect on DPLL. Again, modelling for local search is shown to be distinct from modelling for DPLL.

4 Conclusion

We showed that local search performance on two hard problems can be boosted by several orders of magnitude, simply by remodelling the problems. The aims of our remodelling were to reduce variable dependency chains and to increase solution density, and we believe that these aims should be borne in mind when modelling a problem for solution by local search. They are quite different from the aims of modelling for DPLL, such as symmetry elimination and the level of consistency achieved by unit propagation. Thus modelling for local search is distinct from modelling for DPLL and is worth studying in its own right. Increased solution density might also be expected to aid backtrack search but this is not necessarily true. Structured SAT problems are likely to contain clusters of solutions, and Minton et al.'s *nonsystematic search hypothesis* [21] is that local search may benefit more than backtrack search from high solution density. This is because local search is largely immune to solution clustering, whereas backtrack search may start from a point that is very far from any cluster.

The remodelling approach can be seen as complementary to the preprocessing/algorithmic approaches of [15,23]. We are able to use an off-the-shelf local

search algorithm instead of a more complex new algorithm, and do not incur the runtime overhead of maintaining additional information. On the other hand, we require larger SAT encodings that also incur runtime overheads, and may use a prohibitive amount of memory in some cases. It would be interesting to combine the two approaches, by removing some structure via remodelling then handling what remains via dependency analysis.

Our new modelling techniques should find application to other problems. The parity constraint shallow encoding should be useful on other problems containing both clauses and parity constraints, such as the cryptanalysis problems of [20]. The new cardinality constraint encoding has many potential applications but we have not yet compared it empirically to known encodings. The superparallelism technique used to improve Towers of Hanoi can be applied to STRIPS models of other planning problems. Long-range dependencies based on explanatory frame axioms can be added to planning-as-SAT systems. Bounded model checking has a similar structure to planning and contains parity constraints, so it may also benefit from these techniques.

Finally, recall SAT challenge number six from [29]: to handle variable dependencies in local search. Our results further confirm the importance of this challenge, and show that a powerful alternative to modifying local search heuristics is to reduce or eliminate dependencies by remodelling the problem. In fact avoiding variable dependency by remodelling gives better results than (at least some) attempts to handle dependencies during search. Prevention does seem to be better than cure.

Acknowledgements. This material is based in part upon works supported by the Science Foundation Ireland under Grant No. 00/PI.1/C075. Thanks to the anonymous referees for helpful comments.

References

1. C. Ansótegui, F. Manyà. Mapping Problems With Finite-Domain Variables into Problems With Boolean Variables. *Seventh International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science* vol. 3542, Springer, 2004, pp. 1–15.
2. O. Bailleux, Y. Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. *Ninth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2833, Springer, 2003, pp. 108–122.
3. P. Baumgartner, F. Massacci. The Taming of the (X)OR. *Computational Logic*, 2000.
4. J. M. Crawford, M. J. Kearns, R. E. Shapire. The Minimal Disagreement Parity Problem as a Hard Satisfiability Problem. Technical report, Computational Intelligence Research Laboratory and AT&T Bell Labs, 1994.
5. M. Ernst, T. Millstein, D. S. Weld. Automatic SAT-Compilation of Planning Problems. *Fifteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1997.

6. R. E. Fikes, N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3-4):189–208, 1971.
7. I. P. Gent. Arc Consistency in SAT. *Fifteenth European Conference on Artificial Intelligence*, IOS Press, 2002, pp. 121–125.
8. I. P. Gent, P. Prosser. SAT Encodings of the Stable Marriage Problem With Ties and Incomplete Lists. *Fifth International Symposium on Theory and Applications of Satisfiability Testing*, 2002.
9. I. P. Gent, P. Prosser, B. Smith. A 0/1 Encoding of the GACLex Constraint for Pairs of Vectors. *International Workshop on Modelling and Solving Problems With Constraints*, ECAI, 2002.
10. A. Haas. The Case for Domain-Specific Frame Axioms. *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, F. M. Brown (ed.), Lawrence, KS, 1987. Morgan Kaufmann, 1987.
11. E. A. Hirsch, A. Kojevnikov. Solving Boolean Satisfiability Using Local Search Guided by Unit Clause Elimination. *Seventh International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2239, Springer, 2001, pp. 605–609.
12. F. Hutter, D. A. D. Tompkins, H. H. Hoos. Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT. *Eighth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2470, Springer, 2002, pp. 233–248.
13. S. Kasif. On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks. *Artificial Intelligence* vol. 45, Elsevier, 1990, pp. 275–286.
14. H. Kautz, D. McAllester, B. Selman. Encoding Plans in Propositional Logic. *Fifth International Conference on Principles of Knowledge Representation and Reasoning*, 1996.
15. H. Kautz, D. McAllester, B. Selman. Exploiting Variable Dependency in Local Search. *Poster Sessions of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.
16. H. Kautz, B. Selman. Ten Challenges *Redux*: Recent Progress in Propositional Reasoning and Search. *Ninth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2833, Springer, 2003, pp. 1–18.
17. H. Kautz, B. Selman. Planning as Satisfiability. *Tenth European Conference on Artificial Intelligence*, Wiley, 1992, pp. 359–363.
18. H. Kautz, B. Selman. Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. *National Conference on Artificial Intelligence*, AAAI Press / The MIT Press, 1996, pp. 1194–1201.
19. C. Li. Integrating Equivalence Reasoning into Davis-Putnam Procedure. *Seventeenth National Conference on Artificial Intelligence*, AAAI/MIT Press, 2000, pp. 291–296.
20. F. Massacci. Using Walk-SAT and Rel-SAT for Cryptographic Key Search. *International Joint Conference on Artificial Intelligence*, 1999, pp. 290–295.
21. S. Minton, M. D. Johnston, A. B. Philips, P. Laird. Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58(1-3):161–205, 1992.
22. R. Muhammad, P. J. Stuckey. A Stochastic Non-CNF SAT Solver. *Trends in Artificial Intelligence, Ninth Pacific Rim International Conference on Artificial Intelligence, Lecture Notes in Computer Science* vol. 4099, Springer, 2006, pp. 120–129.

23. D. N. Pham, J. R. Thornton, A. Sattar. Building Structure into Local Search for SAT. *Twentieth International Joint Conference on Artificial Intelligence*, Hyderabad, India, 2007, pp. 2359–2364.
24. S. D. Prestwich. SAT Problems With Chains of Dependent Variables. *Discrete Applied Mathematics* vol. 3037, Elsevier, 2002, pp. 1–22.
25. S. D. Prestwich. Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research* vol. 118, Kluwer Academic Publishers, 2003, pp. 137–150.
26. S. D. Prestwich. Modelling Clique Problems for SAT Local Search. *Third International Workshop on Local Search Techniques in Constraint Satisfaction*, 2006 (to appear).
27. S. D. Prestwich, A. Roli. Symmetry Breaking and Local Search Spaces. *Second International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Lecture Notes in Computer Science* vol. 3524, Springer, 2005, pp. 273-287.
28. B. Selman, personal communication.
29. B. Selman, H. A. Kautz, D. A. McAllester. Ten Challenges in Propositional Reasoning and Search. *Fifteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1997, pp. 50–54.
30. D. A. D. Tompkins, H. H. Hoos. UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. *Seventh International Conference on Theory and Applications of Satisfiability Testing, Lecture Notes in Computer Science* vol. 3542, 2005, pp. 306–320.
31. J. P. Warners. A Linear-Time Transformation of Linear Inequalities Into Conjunctive Normal Form. *Information Processing Letters* 68:63–69, 1998.
32. J. Warners, H. van Maaren. A Two Phase Algorithm for Solving a Class of Hard Satisfiability Problems. *Operations Research Letters* 23(3–5):81–88, 1999.
33. W. Wei, B. Selman. Accelerating Random Walks. *Eighth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2470, Springer, 2002, pp. 216–232.
34. Z. Wu, B. Wah. An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. *Seventeenth National Conference on Artificial Intelligence*, 2000, pp. 310–315.
35. E. Zarpas. Back to the SAT05 Competition: an a Posteriori Analysis of Solver Performance on Industrial Benchmarks. *Journal on Satisfiability, Boolean Modeling and Computation* vol. 2, 2006, research note, pp. 229–237.

Combining Adaptive Noise and Look-Ahead in Local Search for SAT*,**

Chu Min Li[1], Wanxia Wei[2], and Harry Zhang[2]

[1] LaRIA, Université de Picardie Jules Verne
33 Rue St. Leu, 80039 Amiens Cedex 01, France
chu-min.li@u-picardie.fr
[2] Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada, E3B 5A3
{wanxia.wei,hzhang}@unb.ca

Abstract. The adaptive noise mechanism was introduced in $Novelty+$ to automatically adapt noise settings during the search [4]. The local search algorithm G^2WSAT deterministically exploits promising decreasing variables to reduce randomness and consequently the dependence on noise parameters. In this paper, we first integrate the adaptive noise mechanism in G^2WSAT to obtain an algorithm $adaptG^2WSAT$, whose performance suggests that the deterministic exploitation of promising decreasing variables cooperates well with this mechanism. Then, we propose an approach that uses look-ahead for promising decreasing variables to further reinforce this cooperation. We implement this approach in $adaptG^2WSAT$, resulting in a new local search algorithm called $adaptG^2WSAT_P$. Without any manual noise or other parameter tuning, $adaptG^2WSAT_P$ shows generally good performance, compared with G^2WSAT with approximately optimal static noise settings, or is sometimes even better than G^2WSAT. In addition, $adaptG^2WSAT_P$ is favorably compared with state-of-the-art local search algorithms such as $R+adaptNovelty+$ and VW.

1 Introduction

The performance of a $Walksat$ family algorithm crucially depends on noise p and sometimes wp (random walk probability) or dp (diversification probability). For example, it is reported in [9] that running $R\text{-}Novelty$ [9] with $p = 0.4$ instead of $p = 0.6$ degrades its performance by more than 50% for random 3-SAT instances. However, to find the optimal noise settings for each heuristic, extensive experiments on various values of p and sometimes wp or dp are needed because the optimal noise settings vary widely and depend on the types and sizes of the instances.

* A preliminary version of this paper was presented at the 3th International Workshop on LSCS [6], and an extended abstract of this preliminary version will appear in a book, entitled "Trends in Constraint Programming" [7].

** The work of the second author is partially supported by an NSERC (Natural Sciences and Engineering Research Council of Canada) PGS-D scholarship.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 121–133, 2007.

To avoid manual noise tuning, two approaches were proposed. $Auto\text{-}Walksat$ [10] exploits the invariants observed in [9] to estimate the optimal noise settings for an algorithm on a given problem, based on several preliminary unsuccessful runs of the algorithm on this problem. This algorithm then rigorously applies the estimated optimal noise setting to the problem. The adaptive noise mechanism [4] was introduced in $Novelty$+ [3] to automatically adapt noise settings during the search, yielding the algorithm $adaptNovelty$+. This algorithm does not need any manual noise tuning and is effective for a broad range of problems.

One way to diminish the dependence of problem solving on noise settings is to reduce randomness in local search. The local search algorithm G^2WSAT deterministically selects the best promising decreasing variable to flip, if such variables exist [5]. Nevertheless, the performance of G^2WSAT still depends on static noise settings, since when there is no promising decreasing variable, a heuristic, such as $Novelty$++, is used to select a variable to flip, depending on two probabilities, p and dp. Furthermore, G^2WSAT does not favor those flips that will generate promising decreasing variables to minimize its dependence on noise settings.

In this paper, we first incorporate the adaptive noise mechanism of $adaptNovelty$+ in G^2WSAT to obtain an algorithm $adaptG^2WSAT$. Experimental results suggest that the deterministic exploitation of promising decreasing variables in $adaptG^2WSAT$ enhances this mechanism. Then, we integrate a look-ahead approach in $adaptG^2WSAT$ to favor those flips that can generate promising decreasing variables, resulting in a new local search algorithm called $adaptG^2WSAT_P$. Without any manual noise or other parameter tuning, $adaptG^2WSAT_P$ shows generally good performance, compared with G^2WSAT with approximately optimal static noise settings, or is sometimes even better than G^2WSAT. Moreover, $adaptG^2WSAT_P$ compares favorably with state-of-the-art algorithms such as R+$adaptNovelty$+ [1] and VW [11].

2 G^2WSAT and $adaptG^2WSAT$

2.1 G^2WSAT

Given a CNF formula $\mathcal{F}$ and an assignment A, the objective function that local search for SAT attempts to minimize is usually the total number of unsatisfied clauses in $\mathcal{F}$ under A. Let x be a variable. The break of x, $break(x)$, is the number of clauses in $\mathcal{F}$ that are currently satisfied but will be unsatisfied if x is flipped. The make of x, $make(x)$, is the number of clauses in $\mathcal{F}$ that are currently unsatisfied but will be satisfied if x is flipped. The score of x with respect to A, $score_A(x)$, is the improvement of the objective function if x is flipped. The score of x should be the difference between $make(x)$ and $break(x)$. We write $score_A(x)$ as $score(x)$ if A is clear from the context.

Heuristics $Novelty$ [9] and $Novelty$++ [5] select a variable to flip from a randomly selected unsatisfied clause c as follows.

$Novelty(p)$**:** Sort the variables in c by their scores, breaking ties in favor of the least recently flipped variable. Consider the best and second best variables from the sorted variables. If the best variable is not the most recently flipped one in c, then pick it. Otherwise, with probability p, pick the second best variable, and with probability 1-p, pick the best variable.

$Novelty$++**(**p**,** dp**):** With probability dp (diversification probability), pick the least recently flipped variable in c, and with probability 1-dp, do as $Novelty$.

Given a CNF formula $\mathcal{F}$ and an assignment A, a variable x is said to be *decreasing* with respect to A if $score_A(x) > 0$. Promising decreasing variables are defined in [5] as follows:

1. Before any flip, i.e., when A is an initial random assignment, all decreasing variables with respect to A are promising.
2. Let x and y be two different variables and x be not decreasing with respect to A. If, after y is flipped, x becomes decreasing with respect to the new assignment, then x is a promising decreasing variable with respect to the new assignment.
3. A promising decreasing variable remains promising with respect to subsequent assignments in local search until it is no longer decreasing.

G^2WSAT [5] deterministically picks the promising decreasing variable with the highest score to flip, if such variables exist. If there is no promising decreasing variable, G^2WSAT uses a heuristic, such as $Novelty$ [9], $Novelty$+ [3], or $Novelty$++ [5], to pick a variable to flip from a randomly selected unsatisfied clause.

Promising decreasing variables might be considered as the opposite of tabu variables defined in [8,9]; the flips of tabu variables are refused in a number of subsequent steps. Promising decreasing variables are chosen to flip since they probably allow local search to explore new promising regions in the search space, while tabu variables are forbidden since they probably make local search repeat or cancel earlier moves.

2.2 Algorithm $adaptG^2WSAT$

The adaptive noise mechanism [4] in $adaptNovelty$+ can be described as follows. At the beginning of a run, noise p is set to 0. Then, if no improvement in the objective function value has been observed over the last $\theta \times m$ search steps, where m is the number of the clauses of the input formula, and θ is a parameter whose default value in $adaptNovelty$+ is $1/6$, noise p is increased by $p := p + (1 - p) \times \phi$, where ϕ is another parameter whose default value in $adaptNovelty$+ is 0.2. Every time the objective function value is improved, noise p is decreased by $p := p - p \times \phi/2$.

We implement this adaptive noise mechanism of $adaptNovelty$+ in G^2WSAT to obtain an algorithm $adaptG^2WSAT$, and confirm that ϕ and θ need not be tuned for each problem instance or instance type to achieve good performances. That is, like $adaptNovelty$+, $adaptG^2WSAT$ is an algorithm in which no parameter has to be manually tuned to solve a new problem.

2.3 Performances of the Adaptive Noise Mechanism for $adaptG^2WSAT$ and for $adaptNovelty$+

We evaluate the performance of the adaptive noise mechanism for $adaptG^2WSAT$ on 9 groups of benchmark SAT problems.[1] Structured problems come from the SATLIB

[1] All experiments reported in this paper are conducted in Chorus, which consists of 2 dual processor master nodes (Sun V65) with hyperthreading enabled and 80 dual processor compute nodes (Sun V60). Each compute node has two 2.8GHz Intel Xeon processors with 2 to 3 Gigabytes of memory.

repository[2] and Miroslav Velev's SAT Benchmarks.[3] These structured problems include bw_large.c and bw_large.d in Blocksworld, 3bit*31, 3bit*32, e0ddr2*1, e0ddr2*4, enddr2*1, enddr2*8, ewddr2*1, and ewddr2*8 in Beijing, the first 5 instances in Flat200-479, logistics.c and logistics.d in logistics, par16-1, par16-2, par16-3, par16-4, and par16-5 in parity, the 10 satisfiable instances in QG, and all satisfiable formulas in Superscalar Suite 1.0a (SSS.1.0a) except for *bug54.[4] Since these 10 QG instances contain unit clauses, we simplify them using *my_compact*[5] before running every algorithm. Random problems consist of unif04-52, unif04-62, unif04-65, unif04-80, unif04-83, unif04-86, unif04-91, and unif04-99, from the random category in the SAT 2004 competition benchmark.[6] Industrial problems comprise v*1912, v*1915, v*1923, v*1924, v*1944, v*1955, v*1956, and v*1959, from the industrial category in the SAT 2005 competition benchmark.[7]

Table 1 shows the performances of $adaptG^2WSAT$ and G^2WSAT, both using heuristic $Novelty$+, compared with those of $adaptNovelty$+ and $Novelty$+. This table presents the results of these algorithms for only one instance from each group. The random walk probability (wp) is not adjusted and takes the default value 0.01 for the original $Novelty$+, in each algorithm for each instance. G^2WSAT (version 2005) is downloaded from http://www.laria.u-picardie.fr/~cli. $Novelty$+ and $adaptNovelty$+

Table 1. Performance of the adaptive noise mechanism for $adaptG^2WSAT$ using $Novelty+$ and for $adaptNovelty$+. Results in bold indicate the lower degradation in success rate.

algorithm heuristic parameters	cutoff	$Novelty+$ wp=0.01		$adaptNovelty+$ θ=1/6,ϕ=0.2		G^2WSAT $Novelty+$ wp=0.01		$adaptG^2WSAT$ $Novelty+$ θ=1/6,ϕ=0.2	
		p	suc	suc	suc degr	p	suc	suc	suc degr
bw_large.d	10^8	.17	100%	92.80%	7.20%	.20	100%	100%	**0%**
ewddr2*8	10^7	.78	100%	5.20%	94.80%	.52	100%	100%	**0%**
flat200-5	10^8	.54	99.60%	99.20%	0.40%	.60	100%	100%	**0%**
logistics.c	10^5	.41	58.00%	43.20%	25.52%	.52	81.20%	73.20%	**9.85%**
par16-1	10^9	.80	98.00%	42.80%	56.33%	.63	100%	100%	**0%**
qg5-11	10^6	.29	100%	97.20%	**2.80%**	.32	100%	92.40%	7.60%
*bug17	10^7	.82	100%	32.80%	67.20%	.29	66.00%	66.00%	**0%**
unif04-52	10^8	.51	99.60%	94.40%	5.22%	.52	100%	99.20%	**0.80%**
v*1912	10^7	.16	56.00%	50.80%	9.29%	.22	84.00%	81.20%	**3.33%**

are from $UBCSAT$ [13]. The static noise p of G^2WSAT is approximately optimal for G^2WSAT on each instance, and is obtained by comparing $p = 0.10, 0.11, ..., 0.89$, and 0.90 for each instance. The static noise p of $Novelty$+ is different from that of G^2WSAT because $Novelty$+ with its own noise p can perform better than

[2] http://www.satlib.org/

[3] http://www.ece.cmu.edu/∼mvelev/sat_benchmarks.html

[4] The instance *bug54 is hard for every algorithm discussed in this paper.

[5] available at http://www.laria.u-picardie.fr/~cli

[6] http://www.lri.fr/∼simon/contest04/results/

[7] http://www.lri.fr/∼simon/contest/results/

$Novelty$+ with the noise p of G^2WSAT. Each instance is executed 250 times. The success rate of an algorithm for an instance is the number of successful runs divided by 250, and the success rate is intended to be the empirical probability with which the algorithm finds a solution for the instance within the cutoff. For each algorithm on each instance, we report the cutoff ("cutoff") and success rate ("suc"). Let sr be the success rate of G^2WSAT or $Novelty$+ with static noise for an instance, and ar the success rate of $adaptG^2WSAT$ or $adaptNovelty$+ for the same instance. For each instance, we also report the degradation ("suc degr") in success rate of $adaptG^2WSAT$, ((sr-ar)/sr)*100, compared with that of G^2WSAT, and the degradation ("suc degr") in success rate of $adaptNovelty$+, ((sr-ar)/sr)*100, compared with that of $Novelty$+.

According to Table 1, without manual noise tuning, $adaptG^2WSAT$ and $adaptNovelty$+, with the adaptive noise mechanism, achieve good performances, θ and ϕ taking the same fixed values for all problems. Nevertheless, with instance specific noise settings, G^2WSAT and $Novelty$+ achieve success rates the same as or higher than $adaptG^2WSAT$ and $adaptNovelty$+, respectively, for all instances. For all instances except for qg5-11, the degradation in success rate of $adaptG^2WSAT$ compared with that of G^2WSAT is lower than the degradation in success rate of $adaptNovelty$+ compared with that of $Novelty$+. Especially, for bw_large.d, ewddr2*8, par16-1, and *bug17, the degradation in success rate of $adaptG^2WSAT$ compared with that of G^2WSAT is significantly lower than the degradation in success rate of $adaptNovelty$+ compared with that of $Novelty$+.

In Table 1, both $adaptG^2WSAT$ and G^2WSAT use $Novelty$+ to select a variable to flip when there is no promising decreasing variable. Furthermore, $adaptG^2WSAT$ uses the same default values for parameters θ and ϕ as $adaptNovelty$+, to adapt noise. So, it appears that, apart from the implementation details, the only difference between G^2WSAT and $Novelty$+, and between $adaptG^2WSAT$ and $adaptNovelty$+, in Table 1, is the deterministic exploitation of promising decreasing variables in G^2WSAT and $adaptG^2WSAT$. From this table, we observe that the degradation in performance of $adaptG^2WSAT$ compared with that of G^2WSAT is lower than the degradation in performance of $adaptNovelty$+ compared with that of $Novelty$+. This observation suggests that the deterministic exploitation of promising decreasing variables enhances the adaptive noise mechanism. We then expect that better exploitation of promising decreasing variables will further enhance this mechanism.

3 Look-Ahead for Promising Decreasing Variables

3.1 Promising Score of a Variable

Given a CNF formula $\mathcal{F}$ and an assignment A, let x be a variable, let B be obtained from A by flipping x, and let x' be the best promising decreasing variable with respect to B. We define the promising score of x with respect to A as

$$pscore_A(x) = score_A(x) + score_B(x')$$

where $score_A(x)$ is the score of x with respect to A and $score_B(x')$ is the score of x' with respect to B.[8]

[8] x' has the highest $score_B(x')$ among all promising decreasing variables with respect to B.

If there are promising decreasing variables with respect to B, the promising score of x with respect to A represents the improvement in the number of unsatisfied clauses under A by flipping x and then x'. In this case, $pscore_A(x) > score_A(x)$.

If there is no promising decreasing variable with respect to B,

$$pscore_A(x) = score_A(x)$$

since $adaptG^2WSAT$ does not know in advance which variable will be flipped for B (the choice of the variable to flip is made randomly by using $Novelty{+}{+}$).

Given $\mathcal{F}$ and two variables x and y in $\mathcal{F}$, y is said to be a neighbor of x with respect to $\mathcal{F}$ if y occurs in some clause containing x in $\mathcal{F}$. According to Equation 6 in [5], the flipping of x can only change the scores of the neighbors of x. Given an initial assignment, G^2WSAT or $adaptG^2WSAT$ computes the scores for all variables, and then uses Equation 6 in [5] to update the scores of the neighbors of the flipped variable after each step and maintains a list of promising decreasing variables. This update takes time $O(L)$, where L is the upper bound for the sum of the lengths of all clauses containing the flipped variable and is almost a constant for a random 3-SAT problem when the ratio of the number of clauses to the number of variables is a constant. The computation of $pscore_A(x)$ involves the simulation of flipping x and the searching for the largest score of the promising decreasing variables after flipping x. This computation takes time $O(L + \gamma)$, where γ is the upper bound for the number of all the promising decreasing variables in $\mathcal{F}$ after flipping x.

3.2 Integrating Limited Look-Ahead in $adaptG^2WSAT$

We improve $adaptG^2WSAT$ in two ways. The algorithm $adaptG^2WSAT$ maintains a stack, $DecVar$, to store all promising decreasing variables in each step. When there are promising decreasing variables, the improved $adaptG^2WSAT$ chooses the least recently flipped promising decreasing variable among all promising decreasing variables

Function: $Novelty{+}_P(p, wp, c)$

1: **with** probability wp **do** $y \leftarrow$ randomly choose a variable in c;
2: **otherwise**
3: Determine $best$ and $second$, breaking ties in favor of the least recently flipped variable;
/*$best$ and $second$ are the best and second best variables in c according to the scores*/
4: **if** $best$ is the most recently flipped variable in c
5: **then**
6: **with** probability p **do** $y \leftarrow second$;
7: **otherwise** if $pscore(second) >= pscore(best)$ **then** $y \leftarrow second$ **else** $y \leftarrow best$;
8: **else**
9: **if** $best$ is more recently flipped than $second$
10: **then if** $pscore(second) >= pscore(best)$ **then** $y \leftarrow second$ **else** $y \leftarrow best$;
11: **else** $y \leftarrow best$;
12: **return** y;

Fig. 1. Function $Novelty{+}_P$

in $|DecVar|$ to flip. Otherwise, the improved $adaptG^2WSAT$ selects a variable to flip from a randomly chosen unsatisfied clause c, using heuristic $Novelty+_P$ (see Fig. 1), which extends $Novelty+$ [3], to exploit limited look-ahead.

Let $best$ and $second$ denote the best and second best variables respectively, measured by the scores of variables in c. $Novelty+_P$ computes the promising scores for only $best$ and $second$, only when $best$ is more recently flipped than $second$ (including the case in which $best$ is the most recently flipped variable, where the computation is performed with probability $1-p$), in order to favor the less recently flipped $second$. In this case, $score(second) < score(best)$. As is suggested by the success of $HSAT$ [2] and $Novelty$ [9], a less recently flipped variable is generally better if it can improve the objective function at least as well as a more recently flipped variable does. Accordingly, $Novelty+_P$ prefers $second$ if $second$ is less recently flipped than $best$ and if $pscore(second) \geq pscore(best)$.

The improved $adaptG^2WSAT$ is called $adaptG^2WSAT_P$ and is sketched in Fig. 2. Note that wp (random walk probability) is also automatically adjusted and $wp = p/10$. The reason for adjusting wp this way is that, when noise needs to be high, local search should also be well randomized, and when low noise is sufficient, random walks are often not needed. The setting $wp = p/10$ comes from the fact that $p = 0.5$ and $dp = 0.05$ give the best results for random 3-SAT instances in G^2WSAT.

Given a CNF formula $\mathcal{F}$ and an assignment A, the set of assignments obtained by flipping one variable of $\mathcal{F}$ is called the *1-flip neighborhood* of A, and the set of assignments obtained by flipping two variables of $\mathcal{F}$ is called the *2-flip neighborhood* of

Algorithm. $adaptG^2WSAT_P$(SAT-formula $\mathcal{F}$)

```
1:  for try=1 to Maxtries do
2:      A← randomly generated truth assignment; p=0; wp=0;
3:      Store all decreasing variables in stack DecVar;
4:      for flip=1 to Maxsteps do
5:          if A satisfies F then return A;
6:          if |DecVar| > 0
7:          then
8:              y←the least recently flipped promising decreasing variable among
9:                  all promising decreasing variables in |DecVar|;
10:         else
11:             c←randomly selected unsatisfied clause under A;
12:             y ← Novelty+_P(p, wp, c);
13:         A ← A with y flipped;
14:         Adapt p and wp;
15:         Delete variables that are no longer decreasing from DecVar;
16:         Push new decreasing variables into DecVar which are different from
17:          y and were not decreasing before y is flipped;
18: return Solution not found;
```

Fig. 2. Algorithm $adaptG^2WSAT_P$

A. The algorithm $adaptG^2WSAT_P$ exploits only the 1-flip neighborhoods, since the limited look-ahead is just used as a heuristic to select the next variable to flip.

We find that in $adaptG^2WSAT$ and $adaptG^2WSAT_P$, which use heuristics $Novelty$++ and $Novelty$+$_P$, respectively, $\theta = 1/5$ and $\phi = 0.1$ give slightly better results on the 9 groups of instances presented in Section 2.3 than $\theta = 1/6$ and $\phi = 0.2$, their original default values in $adaptNovelty$+. So, in $adaptG^2WSAT_P$, $\theta = 1/5$ and $\phi = 0.1$.

In this paper, $adaptG^2WSAT_P$ is improved in two ways, based on the preliminary $adaptG^2WSAT_P$ described in the preliminary version of this paper [6,7]. The first improvement is that, when promising decreasing variables exist, $adaptG^2WSAT_P$ no longer computes the promising scores for the δ promising decreasing variables with higher scores in $|DecVar|$, where δ is a parameter, but chooses the least recently flipped promising decreasing variable among all promising decreasing variables in $|DecVar|$ to flip. As a result, $adaptG^2WSAT_P$ no longer needs parameter δ. The reasons for this first improvement are that, usually the scores of promising decreasing variables are close and so such variables can improve the objective function roughly the same, and that flipping the least recently flipped promising decreasing variable can increase the mobility and coverage [12] of a local search algorithm in the search space. The second improvement is that, when there is no promising decreasing variable, $adaptG^2WSAT_P$ uses $Novelty$+$_P$ instead of $Novelty$++$_P$ [6,7], to select a variable to flip from a randomly chosen unsatisfied clause c. The difference between $Novelty$+$_P$ and $Novelty$++$_P$ is that, with wp (random walk probability), $Novelty$+$_P$ randomly chooses a variable to flip from c, but with dp (diversification probability), $Novelty$++$_P$ chooses a variable in c, whose flip will falsify the least recently satisfied clause. Considering that $adaptG^2WSAT_P$ deterministically uses both promising decreasing variables and promising scores, adding a small amount of randomness[9] to the search may help find a solution.

4 Evaluation

We evaluate $adaptG^2WSAT_P$ on the 9 groups of instances, or the 56 instances, presented in Section 2.3. For an instance and an algorithm, we report the median flip number ("#flips") and the median run time ("time") in seconds, for this algorithm to find a solution for this instance. Each instance is executed 250 times. If an algorithm can successfully find a solution for an instance in at least 126 runs, the median flip number and median run time are calculated based on these 250 runs. If an algorithm cannot achieve a success rate greater than 50% on an instance even if the cutoff is greater than or equal to the maximum value among the cutoffs of all other algorithms, the median flip number and median run time cannot be calculated; we use "$> Maxsteps$" (greater than $Maxsteps$) to denote the median flip number and use "n/a" to denote the median run time, where $Maxsteps$ is the cutoff for this algorithm on this instance. If the median flip number and median run time of G^2WSAT with any noise settings for an instance cannot be calculated, we also use n/a to denote the optimal noise setting. Results in bold indicate the best performance for an instance.

[9] In general, wp ranges from 0% to 10%.

4.1 Comparison of Performances of $adaptG^2WSAT_P$, G^2WSAT, and $adaptG^2WSAT$

We compare the performances of $adaptG^2WSAT_P$, G^2WSAT with approximately optimal noise settings, and $adaptG^2WSAT$ in Table 2, where $adaptG^2WSAT_P$ uses $Novelty+_P$, and G^2WSAT and $adaptG^2WSAT$ use $Novelty++$, to pick a variable to flip, when there is no promising decreasing variable. On the instances that G^2WSAT can solve in reasonable time, except for qg7-13, the performance of $adaptG^2WSAT_P$ is comparable to that of G^2WSAT with approximately optimal noise settings. Moreover, $adaptG^2WSAT_P$ can solve 3bit*31, 3bit*32, *bug5, *bug38, *bug39, and *bug40, which are hard for G^2WSAT with any static noise settings. More importantly, $adaptG^2WSAT_P$ does not need any manual tuning of p and wp for each instance while G^2WSAT needs manual tuning of p and dp for each instance. In other words, G^2WSAT cannot achieve the performance shown in this table by using the same p and dp for the broad range of instances.

On the instances that $adaptG^2WSAT$ can solve in reasonable time, the performance of $adaptG^2WSAT_P$ is comparable to that of $adaptG^2WSAT$. Furthermore, $adaptG^2WSAT_P$ can solve 3bit*31, 3bit*32, *bug5, *bug38, *bug39, and *bug40, which are hard for $adaptG^2WSAT$. In addition, among the 56 instances presented in this table, $adaptG^2WSAT_P$ exhibits the best run time performance and/or the best flip number performance on the 13 instances among $adaptG^2WSAT_P$, G^2WSAT with approximately optimal noise settings, and $adaptG^2WSAT$, while $adaptG^2WSAT$ is never the best.

4.2 Comparison of Performances of $adaptG^2WSAT_P$, $R+adaptNovelty+$, and VW

$R+adaptNovelty+$ is $adaptNovelty+$ with preprocessing to add a set of resolvents of length ≤ 3 into the input formula [1]. VW [11] is an extension of $Walksat$. VW adjusts and smoothes variable weights, and takes variable weights into account when selecting a variable to flip. $R+adaptNovelty+$, G^2WSAT with p=0.50 and dp=0.05, and VW won the gold, silver, and bronze medals, respectively, in the satisfiable random formula category in the SAT 2005 competition.[10]

Table 3 compares the performance of $adaptG^2WSAT_P$ with the performances of $R+adaptNovelty+$ and VW. We download $R+adaptNovelty+$ and VW from http://www.satcompetition.org/. We use the default value 0.01 for the random walk probability in $R+adaptNovelty+$, when running this algorithm. In this table, instances with † on the right constitute the entire set of instances that were used to originally evaluate $R+adaptNovelty+$ in [1]. Among the 56 instances presented in this table, in terms of run time, $adaptG^2WSAT_P$, $R+adaptNovelty+$, and VW are the best algorithms on the 32, 16, and 13 instances, respectively. Also, among the 56 instances, in terms of run time, $adaptG^2WSAT_P$ outperforms $R+adaptNovelty+$ and VW on the 38 and 42 instances, respectively.

[10] http://www.satcompetition.org/

Table 2. Performance of $adaptG^2WSAT_P$, $adaptG^2WSAT$, and G^2WSAT with approximately optimal noise settings

	$adaptG^2WSAT_P$		$adaptG^2WSAT$		G^2WSAT		
	#flips	time	#flips	time	optimal	#flips	time
bw_large.c	**1083947**	**3.650**	3553694	10.175	(.21, 0)	2119497	3.699
bw_large.d	**1542898**	8.590	9626411	49.635	(.16, 0)	3237895	**7.180**
3bit*31	**87158**	**0.780**	$> 10^7$	n/a	n/a	$> 10^7$	n/a
3bit*32	**60518**	**0.565**	$> 10^7$	n/a	n/a	$> 10^7$	n/a
e0ddr2*1	4520164	19.275	831073	2.595	(.14, .09)	**254182**	**0.910**
e0ddr2*4	641587	2.855	208815	0.805	(.23, .1)	**117266**	**0.540**
enddr2*1	982540	4.570	153905	0.640	(.18, .1)	**97451**	**0.535**
enddr2*8	412624	2.385	135332	0.585	(.16, .09)	**90076**	**0.480**
ewddr2*1	492907	2.470	137430	0.600	(.18, .1)	**89420**	**0.505**
ewddr2*8	262177	1.385	116917	0.535	(.16, .1)	**67854**	**0.425**
flat200-1	36764	0.025	42053	0.020	(.49, .08)	**25358**	**0.010**
flat200-2	288521	0.160	303515	0.135	(.49, .07)	**171487**	**0.085**
flat200-3	71324	0.045	89515	0.040	(.51, .05)	**51037**	**0.025**
flat200-4	314273	0.180	323353	0.145	(.49, .05)	**178842**	**0.095**
flat200-5	4963846	2.675	4173580	1.810	(.49, .08)	**3008035**	**1.455**
logistics.c	54777	0.075	46875	0.060	(.24, .07)	**38177**	**0.040**
logistics.d	83894	0.185	102575	0.165	(.2, .08)	**78013**	**0.105**
par16-1	58937999	27.955	76985828	29.870	(.51, .01)	**48342381**	**20.835**
par16-2	130634181	64.300	140615726	57.170	(.59, .01)	**73324801**	**32.460**
par16-3	104764223	50.865	112297525	44.885	(.58, .01)	**80700698**	**33.223**
par16-4	133899858	63.595	174053106	68.735	(.5, .02)	**89662042**	**39.256**
par16-5	124873168	59.865	133250726	53.385	(.54, .02)	**83818097**	**35.688**
qg1-07	6413	0.025	7370	0.020	(.38, 0)	**4599**	**0.010**
qg1-08	361229	4.740	448660	3.635	(.11, .03)	**339312**	**1.350**
qg2-07	3869	0.020	4708	0.025	(.33, .01)	**2648**	**0.005**
qg2-08	**1262398**	8.960	1473258	9.565	(.22, 0)	1449931	**6.270**
qg3-08	36322	0.125	36046	0.040	(.44, .05)	**20517**	**0.015**
qg4-09	68472	0.310	70659	0.100	(.37, 0)	**48741**	**0.075**
qg5-11	20598	0.210	23431	0.275	(.38, .01)	**12559**	**0.080**
qg6-09	414	0.005	441	0.005	(.41, .08)	**340**	**0.000**
qg7-09	392	0.005	318	0.005	(.41, .1)	**316**	**0.015**
qg7-13	$> 10^8$	n/a	$> 10^8$	n/a	(.33, 0)	**4768987**	**50.809**
*bug3	$> 10^8$	n/a	$> 10^8$	n/a	n/a	$> 10^8$	n/a
*bug4	$> 10^8$	n/a	$> 10^8$	n/a	n/a	$> 10^8$	n/a
*bug5	**1460519**	**6.050**	$> 10^8$	n/a	n/a	$> 10^8$	n/a
*bug17	107501	**1.170**	425730	5.130	(.15, .15)	**63582**	1.355
*bug38	**181666**	**0.745**	$> 10^8$	n/a	n/a	$> 10^8$	n/a
*bug39	**75743**	**0.390**	$> 10^8$	n/a	n/a	$> 10^8$	n/a
*bug40	**182279**	**0.890**	$> 10^8$	n/a	n/a	$> 10^8$	n/a
*bug59	102853	1.080	268332	2.475	(.62, .06)	**52276**	**0.408**
unif04-52	5588325	6.065	6763462	5.570	(.4, .07)	**4991465**	**4.295**
unif04-62	530432	0.590	768215	0.640	(.49, .03)	**386031**	**0.335**
unif04-65	1406786	1.560	1566427	1.315	(.48, .06)	**1289658**	**0.918**
unif04-80	3059121	3.575	3751125	3.300	(.45, .1)	**1908125**	**1.760**
unif04-83	8370126	9.930	6589739	5.860	(.43, .09)	**4370302**	**3.112**
unif04-86	6288398	7.450	5817258	5.250	(.43, .09)	**3429233**	**2.442**
unif04-91	659313	0.780	789717	0.730	(.5, .05)	**414399**	**0.324**
unif04-99	**4054201**	4.985	7746102	7.205	(.45, .02)	4931360	**4.530**
v*1912	**3454184**	84.115	3683237	78.625	(.16, 0)	3554771	**65.509**
v*1915	12928287	409.480	14636382	328.450	(.19, .02)	**12510065**	**288.966**
v*1923	1200896	25.030	1358055	16.630	(.42, 0)	**1065848**	**13.386**
v*1924	**1389813**	28.040	1756779	29.855	(.21, .04)	1613496	**23.019**
v*1944	4248279	216.700	4386535	156.67	(.20, 0)	**3667138**	**126.398**
v*1955	1404357	56.240	1417356	32.195	(.29, .01)	**1152386**	**28.669**
v*1956	1762589	71.100	1849539	68.365	(.26, .02)	**1599232**	**46.434**
v*1959	612589	27.985	786925	32.815	(.37, .01)	**498563**	**16.276**

Table 3. Experimental results for R+$adaptNovelty$+, $adaptG^2WSAT_P$, and VW

	$R{+}adaptNovelty{+}$		$adaptG^2WSAT_P$		VW	
	#flips	time	#flips	time	#flips	time
bw_large.c†	9489817	29.140	**1083947**	**3.650**	1868393	5.960
bw_large.d	27179763	152.160	**1542898**	**8.590**	2963500	18.120
3bit*31	152565	1.645	87158	0.780	**37487**	**0.290**
3bit*32	133945	1.640	60518	0.565	**21858**	**0.160**
e0ddr2*1†	**2488226**	**10.630**	4520164	19.275	6549282	22.530
e0ddr2*4†	**355044**	**1.530**	641587	2.855	1894243	7.850
enddr2*1†	**331420**	**1.555**	982540	4.570	4484178	17.605
enddr2*8†	**11753**	**0.020**	412624	2.385	3493986	15.505
ewddr2*1†	**154825**	**0.675**	492907	2.470	4714786	18.410
ewddr2*8†	**32527**	**0.100**	262177	1.385	4956356	21.785
flat200-1	50600	0.030	**36764**	**0.025**	187053	0.085
flat200-2	535300	0.280	**288521**	**0.160**	1318485	0.650
flat200-3	161169	0.085	**71324**	**0.045**	664550	0.330
flat200-4	577180	0.290	**314273**	**0.180**	2747696	1.345
flat200-5	15841761	8.366	**4963846**	**2.675**	26137279	13.119
logistics.c†	57693	**0.075**	**54777**	**0.075**	70446	0.085
logistics.d	162737	0.220	**83894**	**0.185**	340379	0.395
par16-1†	80339283	37.645	**58937999**	**27.955**	$> 10^9$	n/a
par16-2†	324826713	157.455	**130634181**	**64.300**	$> 10^9$	n/a
par16-3†	224140856	107.410	**104764223**	**50.865**	$> 10^9$	n/a
par16-4†	274054172	129.660	**133899858**	**63.595**	$> 10^9$	n/a
par16-5†	264871971	125.025	**124873168**	**59.865**	$> 10^9$	n/a
qg1-07†	9882	**0.015**	**6413**	0.025	21304	0.055
qg1-08†	676122	**2.300**	**361229**	4.740	2548200	69.325
qg2-07†	6147	**0.010**	**3869**	0.020	9181	0.035
qg2-08†	2200276	**8.440**	**1262398**	8.960	8843525	277.735
qg3-08†	53998	**0.070**	**36322**	0.125	137354	0.185
qg4-09†	105386	**0.165**	**68472**	0.310	264297	0.505
qg5-11†	36856	0.215	**20598**	**0.210**	39907	0.410
qg6-09†	542	**0.000**	**414**	**0.000**	1014	**0.000**
qg7-09†	531	**0.000**	**392**	**0.000**	1037	**0.000**
qg7-13†	**5113772**	**66.680**	$> 10^8$	n/a	8843466	307.620
*bug3	62148492	360.920	$> 10^8$	n/a	**1974994**	**4.875**
*bug4	$> 10^8$	n/a	$> 10^8$	n/a	**177511**	**0.460**
*bug5	66283256	431.395	1460519	6.050	**280071**	**0.735**
*bug17	6020734	141.875	107501	1.170	**32999**	**0.275**
*bug38	4699436	32.735	181666	0.745	**157834**	**0.385**
*bug39	9693455	54.345	**75743**	0.390	83287	**0.220**
*bug40	17465338	125.010	182279	0.890	**98834**	**0.290**
*bug59	389865	4.150	102853	1.080	**66090**	**0.345**
unif04-52†	24720067	21.335	**5588325**	**6.065**	22594215	17.115
unif04-62†	1484946	1.280	**530432**	**0.590**	3321105	2.605
unif04-65†	9043996	7.885	**1406786**	**1.560**	4505318	3.520
unif04-80†	5432957	4.780	**3059121**	**3.575**	20083928	16.515
unif04-83†	291310536	255.685	**8370126**	**9.930**	25897048	21.590
unif04-86†	38667651	34.045	**6288398**	7.450	8536496	**7.170**
unif04-91†	1581843	1.370	**659313**	**0.780**	3097695	2.725
unif04-99†	16856278	14.850	**4054201**	**4.985**	17422353	15.400
v*1912	6812718	148.735	**3454184**	**84.115**	61152892	3037.695
v*1915	78909897	2208.900	**12928287**	**409.480**	$> 10^8$	n/a
v*1923	2736569	51.662	**1200896**	**25.030**	9820793	340.430
v*1924	2931225	60.319	**1389813**	**28.040**	13744232	515.720
v*1944	6153990	373.905	**4248279**	**216.700**	58541545	7971.731
v*1955	2755333	89.455	**1404357**	**56.240**	10396220	1073.960
v*1956	2865074	114.685	**1762589**	**71.100**	13419375	1437.035
v*1959	2420412	118.335	**612589**	**27.985**	11433482	1377.245

Table 4. Experimental results for the preliminary $adaptG^2WSAT_P$ and $adaptG^2WSAT_P$

	preliminary $adaptG^2WSAT_P$		$adaptG^2WSAT_P$	
	#flips	time	#flips	time
*bug5	$> 10^8$	n/a	**1460519**	**6.050**
*bug17	133691	2.820	**107501**	**1.170**
*bug38	$> 10^8$	n/a	**181666**	**0.745**
*bug39	$> 10^8$	n/a	**75743**	**0.390**
*bug40	$> 10^8$	n/a	**182279**	**0.890**
*bug59	179091	4.965	**102853**	**1.080**

4.3 Comparison of Performances of $adaptG^2WSAT_P$ and Preliminary $adaptG^2WSAT_P$

Our experimental results show that $adaptG^2WSAT_P$ exhibits better performance than the preliminary $adaptG^2WSAT_P$ on some instances from SSS.1.0a presented in Section 2.3. According to our experimental results, on the remaining instances presented in Section 2.3, the overall performance of $adaptG^2WSAT_P$ is close to that of the preliminary $adaptG^2WSAT_P$. Table 4 indicates that $adaptG^2WSAT_P$ exhibits good performance on the 6 instances from SSS.1.0a while the preliminary $adaptG^2WSAT_P$ has difficulty on 4 out of these 6.

5 Conclusion

We have found that the deterministic exploitation of promising decreasing variables can enhance the adaptive noise mechanism in local search for SAT, and thus integrated this adaptive noise mechanism in G^2WSAT to obtain the algorithm $adaptG^2WSAT$. We then have proposed a limited look-ahead approach to favor those flips generating promising decreasing variables to further improve the adaptive noise mechanism. The look-ahead approach is based on the promising scores of variables, meaning that after flipping a variable x, the score of the best promising decreasing variable should be added to the score of x to improve the objective function. The resulting algorithm is called $adaptG^2WSAT_P$.

There are two new parameters in $adaptG^2WSAT_P$, θ and ϕ, which are from $adaptNovelty$+ and are used to implement the adaptive noise mechanism. However, noise p and random walk probability wp are entirely automatically adapted. Our experimental results confirm that, like θ and ϕ in $adaptNovelty$+, θ and ϕ in $adaptG^2WSAT_P$ are substantially less sensitive to problem instances and problem types than are p and wp [4], and our results also show that the same fixed default values of θ and ϕ allow $adaptG^2WSAT_P$ to achieve good performances for a broad range of SAT problems. Moreover, our experimental results show that, without any manual noise or other parameter tuning, $adaptG^2WSAT_P$ shows generally good performance, compared with G^2WSAT with approximately optimal static noise settings, or is sometimes even better than G^2WSAT, and that $adaptG^2WSAT_P$ compares favorably with state-of-the-art algorithms such as R+$adaptNovelty$+ and VW.

We plan to optimize the computation of promising scores, which actually is not incremental. In addition, the efficient implementation techniques of $UBCSAT$, the variable weight smoothing technique proposed in VW, and the preprocessing used in R+$adaptNovelty$+ could be integrated into $adaptG^2WSAT_P$.

References

1. Anbulagan, D. N. Pham, J. Slaney, and A. Sattar. Old Resolution Meets Modern SLS. In *Proceedings of AAAI-2005*, pages 354–359. AAAI Press, 2005.
2. I. P. Gent and T. Walsh. Towards an Understanding of Hill-Climbing Procedures for SAT. In *Proceedings of AAAI-1993*, pages 28–33. AAAI Press, 1993.
3. H. Hoos. On the Run-Time Behavior of Stochastic Local Search Algorithms for SAT. In *Proceedings of AAAI-1999*, pages 661–666. AAAI Press, 1999.
4. H. Hoos. An Adaptive Noise Mechanism for WalkSAT. In *Proceedings of AAAI-2002*, pages 655–660. AAAI Press, 2002.
5. C. M. Li and W. Q. Huang. Diversification and Determinism in Local Search for Satisfiability. In *Proceedings of SAT-2005*, pages 158–172. Springer, LNCS 3569, 2005.
6. C. M. Li, W. Wei, and H. Zhang. Combining Adaptive Noise and Look-Ahead in Local Search for SAT. In *Proceedings of LSCS-2006*, pages 2–16, 2006.
7. C. M. Li, W. Wei, and H. Zhang. Combining Adaptive Noise and Look-Ahead in Local Search for SAT. In F. Benhamou, N. Jussien, and B. O'Sullivan, editors, *Trends in Constraint Programming*, chapter 2. Hermes Science, 2007 (to appear).
8. B. Mazure, L. Sais, and E. Gregoire. Tabu Search for SAT. In *Proceedings of AAAI-1997*, pages 281–285. AAAI Press, 1997.
9. D. A. McAllester, B. Selman, and H. Kautz. Evidence for Invariant in Local Search. In *Proceedings of AAAI-1997*, pages 321–326. AAAI Press, 1997.
10. D. J. Patterson and H. Kautz. Auto-Walksat: A Self-Tuning Implementation of Walksat. Electronic Notes on Discrete Mathematics 9, 2001.
11. S. Prestwich. Random Walk with Continuously Smoothed Variable Weights. In *Proceedings of SAT-2005*, pages 203–215. Springer, LNCS 3569, 2005.
12. D. Schuurmans and F. Southey. Local Search Characteristics of Incomplete SAT Procedures. In *Proceedings of AAAI-2000*, pages 297–302. AAAI Press, 2000.
13. D. A. D. Tompkins and H. H. Hoos. UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. In *Proceedings of SAT-2004*, pages 306–315. Springer, LNCS 3542, 2004.

From Idempotent Generalized Boolean Assignments to Multi-bit Search

Marijn Heule* and Hans van Maaren

Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Sciences
Delft University of Technology
marijn@heule.nl, h.vanmaaren@tudelft.nl

Abstract. This paper shows that idempotents in finite rings of integers can act as Generalized Boolean Assignments (GBA's) by providing a completeness theorem. We introduce the notion of a generic Generalized Boolean Assignment. The mere propagation of such an assignment reveals feasibility (existence of a solution) of a formula in propositional logic. Then, we demystify this general concept by formulating the process on the bit-level: It turns out that propagation of a GBA only simulates bitwise (non-communicating) parallel computing. We capitalize on this by modifying the state-of-the-art local search SAT solver UnitWalk accordingly. This modification involves a more complicated parallelism.

1 Introduction

Propositional Logic and Elementary Arithmetic are - in some sense - similar systems. We provide additional evidence of this by introducing Generalized Boolean Models (GBM's) as certain sets of idempotents in finite residue class rings of integers, and a completeness theorem. We also offer a construction of so-called generic Generalized Boolean Assignments (generic GBA's). We show that formula feasibility can be checked by evaluating its "truth" value under one single generic assignment. These modeling possibilities feature an attractive mathematical simplicity. However, analysis of the proof of the completeness theorem and the process of constructing generic GBA's shows that the above modeling possibility only simulates (non-communicating) parallel computing.

Current Satisfiability (SAT) solvers do not use the opportunity of a k-bit processor to simulate parallel 1-bit (Boolean) search on k 1-bit processors. Conventional parallel SAT solving [3,4,9] differs from the proposed method in section 3: The former realizes performance gain by dividing the workload over multiple processors and some minor changes to the solving algorithm, while the latter uses a single processor and requires significant modifications to the algorithm.

SAT solvers that use multi-bit heuristics frequently (counters for instance), are not very suitable for modification in this respect. However, SAT solvers whose

* Supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.306.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 134–147, 2007.

computational "center of gravity" consists of propagating truth values (or other 1-bit operations) may profit from this opportunity. One of such is the state-of-the-art local search SAT solver UnitWalk [6]. We show that UnitWalk can be upgraded using a single k-bit processor. This results in a considerable speed-up.

2 Idempotents and Generalized Boolean Assignments

The concepts in this section could have been cast into the format of Boolean Algebras [5]. As such, we do not claim that the ideas and results are completely new. However, using a little bit of elementary number theory it is possible to directly relate the concepts needed to familiar arithmetical operations. We preferred to do the latter. On the other hand, to understand the essentials of the next sections, it does not hurt the reader much to continue reading from **Back to Booleans** at the end of this section.

An *idempotent* x in the ring of integers modulo m is an element satisfying $x^2 \equiv x$ (modulo m). For given m, a *Generalized Boolean Model* (GBM) $\mathcal{I}$ is a set of idempotents modulo m obeying the three closure rules:

- $0, 1 \in \mathcal{I}$;
- If $x \in \mathcal{I}$ then $1 - x \in \mathcal{I}$. Notice that $(1 - x)^2 \equiv 1 - x$ (modulo m).
- If $x, y \in \mathcal{I}$ then $xy \in \mathcal{I}$. Notice that $(xy)^2 \equiv xy$ (modulo m).

Given a formula $\mathcal{F}$ in Propositional Logic a *Generalized Boolean Assignment* (GBA) is a mapping from its set of variables to a GBM $\mathcal{I}$. Evaluating the "truth" value of $\mathcal{F}$ under a GBA simply follows the rule of translating $\neg x$ by the arithmetic operation 1 - *value*(x) and conjunction $x \wedge y$ by the operation *value*(x)·*value*(y) (both modulo m), recursively.

Example 1. Consider Z_6, the ring of integers modulo 6. Idempotents modulo 6 are 0, 1, 3 and 4. Let $\mathcal{F}$ be the formula

$$\neg(x \rightarrow (y \vee (x \wedge z))) \tag{1}$$

which is equivalent to

$$x \wedge (\neg y \wedge \neg(x \wedge z)) \tag{2}$$

and assigning $x := 3$, $y := 4$ and $z := 1$, we calculate

$$3 \times ((1 - 4) \times (1 - (3 \times 1))) \equiv 0 \text{ (modulo 6)} \tag{3}$$

By assigning $x := 3$, $y := 4$ and $z := 0$ however, $\mathcal{F}$ evaluates to the value 3, as the reader may verify. Following from the above, each evaluation of a formula under a GBA results in an idempotent in $\mathcal{I}$, due to the closure rules posed on $\mathcal{I}$.

Example 2. Again consider Z_6 and the formula $x \wedge y$. The reader may check that there are 16 possible GBA's of which 7 evaluate to a non-zero idempotent.

Drawing GBA's randomly, the probability of hitting a non-zero idempotent outcome is $\frac{7}{16}$, while in the standard Boolean situation this probability is $\frac{1}{4}$.

The above example shows that random sampling GBA's (or, equivalently, multi-bit Boolean patterns) may hit solutions earlier, in about the same time. As such this is done in [7], where Boolean "patterns" (rather than Booleans) are propagated through a circuit in the order to increase the probability of hitting a solution - indicating an error in their application.

Although this random sampling can be viewed as a rather straight forward parallelism, we claim that to perform efficient multi-bit propagation for SAT solving is not straight forward at all: In [7] at each step, variables are either unassigned or assigned a *full* Boolean pattern, while in the proposed propagation variables can also be assigned a *partial* Boolean assignment.

Theorem 1 (Completeness Theorem). If $\mathcal{F}$ is a formula and $\mathcal{I}$ a Generalized Boolean Model, $\mathcal{F}$ is Satisfiable if and only if there exists a GBA under which $\mathcal{F}$ evaluates to a non-zero idempotent.

Proof: *If $\mathcal{F}$ is Satisfiable, then a $\{0,1\}$-assignment exists under which $\mathcal{F}$ evaluates to 1. This evaluation remains valid in each $\mathcal{I}$. If $\mathcal{F}$ evaluates to a non-zero idempotent w modulo m, there must be a prime factor of m, say p, such that w is non-zero modulo p. Modulo a prime however, the only existing idempotents are 0 and 1, since $x^2 \equiv x$ (modulo p) reduces to $x \equiv 0$ or $x \equiv 1$ (modulo p). Under these circumstances there must be a prime number p which reduces, when calculating modulo p, the GBA to a simple Boolean assignment that satisfies $\mathcal{F}$.*

Construction of generic GBM's. GBM's are constructed as follows: Let m be the product of the first k primes. Let A be the product of a subset of these primes and B the product of the complementary subset. Since A and B are relatively prime, integers r and s exists such that

$$rA + sB = 1 \tag{4}$$

Set $x \equiv rA$ (modulo m). Then $1 - x \equiv sB$ (modulo m) and thus $x(1-x) \equiv 0$ (modulo m). The above observation shows that precisely 2^k different idempotents modulo m exist.

Generic GBA's. Let $\mathcal{F}$ be a formula on n variables and m be the product of the first 2^n primes. As we have seen above, there are 2^{2^n} different idempotents modulo m. This is the same amount as the number of logically independent Boolean functions on n variables. In fact, it is not hard to demonstrate that in the above situation a GBA to the variables exists such that each formula on n variables evaluates to its associated idempotent modulo m, each idempotent representing an equivalence class of Boolean functions. For example:

Example 3. Consider the Boolean functions with $n = 2$, $m = 2 \cdot 3 \cdot 5 \cdot 7 = 210$. The full GBM is $\{0, 1, 15, 21, 36, 70, 85, 91, 105, 106, 120, 126, 141, 175, 190, 196\}$.

A generic GBA is for instance $x = 15$, $y = 21$. In this case $x \wedge \neg y$ evaluates to 120, $\neg(\neg x \wedge y)$ to 85, $x \Leftrightarrow y$ to 175 and $\neg(x \Leftrightarrow y)$ to 36. In this case, every formula on 2 variables can be checked on feasibility by propagating the values $x = 15$ and $y = 21$, and only the outcome 0 (modulo 210) reflects a contradiction. That $(15, 21)$ is generic follows from the fact that $(15, 21)$ is $(1, 1)$ modulo 2, $(0, 0)$ modulo 3, $(0, 1)$ modulo 5 and $(1, 0)$ modulo 7. Notice that - in some sense - we are just doing ordinary SAT in the exponents of the prime factors involved.

Working with GBA's could be beneficial in situations when the arithmetic operations involved can be performed in a small number of clock cycles. More specifically: If we have a 32-bit processor available, formulas with up to 5 variables can be resolved in one propagation run using generic GBA's in about the same time an ordinary Boolean assignment is propagated.

Back to Booleans. Despite the arithmetic elegance of generic GBM's in their capability of representing Boolean functions, it is clear that on the level of implementation integers are not a very welcome ingredient. In fact, the processes explained above are even easier to understand if we return to the Boolean level. To see this, consider the case of functions on 3 variables and the following table:

$$\begin{array}{lll} x & := & \texttt{0 1 0 0 1 1 0 1} \\ y & := & \texttt{0 0 1 0 1 0 1 1} \\ z & := & \texttt{0 0 0 1 0 1 1 1} \end{array}$$

Consider the rows as 8 parallel Boolean assignments to the individual variables (using 8 1-bit processors). We refer to such an assignment as a *multi-bit assignment* (MBA). Notice that the 8 different columns represent the 8 different Boolean assignments in total. Therefore, the above MBA is a *generic MBA* - analogue to generic GBA's. Having an 8-bit processor at our disposal the evaluation of the and-gate $x \wedge y$ results in $\texttt{01001101} \wedge \texttt{00101011} = \texttt{00001001}$, an operation performed in one clock cycle. Bits 5 and 8 certify feasibility. In general, any formula on n variables - based on the primitive operations `AND` ($\wedge$) and `NOT` ($\neg$) - can be resolved, using a generic MBA of 2^n bits (and a k-bit processor with $2^n \leq k$), in as many clock cycles as there are logical operators to perform. The formula is Satisfiable if and only if in the end there is at least one bit equal to 1. In terms of generic GBM's the above 8-bit example would involve calculating modulo $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 = 9699690$.

3 Multi-bit Unit Propagation

This section describes the use of MBA's to parallelize a SAT solving algorithm. However, this differs from conventional parallelism: Modifications of MBA's can be processed in parallel, while, for instance, operations on counters cannot. In general, only 1-bit operations can be parallelized. Therefore, algorithms that potentially benefit from MBA's should have their computational "center of gravity" on assignment modifications.

A widely used procedure for assignment modifications is *unit propagation*: Given a formula $\mathcal{F}$ and an assignment φ. If φ applied to $\mathcal{F}$ (denoted by $\varphi \circ \mathcal{F}$) contains *unit clauses* (clauses of size 1) then the remaining literal in each unit clause is forced to be true - thereby expanding φ. This procedure continues until there are no unit clauses in $\varphi \circ \mathcal{F}$. This section describes a SAT solving algorithm that uses unit propagation at its computational "center of gravity".

The UnitWalk algorithm. For a possible application we focused on local search (incomplete) SAT solvers. In contrast to complete SAT solvers, they are less complicated and work with full assignments. A generic structure of local search SAT solvers is as follows: An assignment φ is generated, earmarking a random Boolean value to all variables. By flipping the truth values of variables, φ can be modified to satisfy as many clauses as possible of the formula at hand. If after a multitude of flips φ still does not satisfy the formula, a new random assignment is generated.

Most local search SAT solvers use counting heuristics to flip the truth value of the variables in a turn-based manner. These heuristics appear hard to parallelize on a single processor. However, the UNITWALK algorithm [6] is an exception. Instead of counting heuristics, it uses unit propagation to flip variables. The UnitWalk SAT solver - based on this algorithm - is the fastest local search SAT solver on many structured instances and won the SAT 2003 competition in the category *All random SAT* [2].

The UNITWALK algorithm (see algorithm 1) flips variables in so-called *periods*: Each period starts with an initial assignment (referred to as master assignment φ_{master}), an empty assignment φ_{active} and an ordering of the variables π. First, unit propagation is executed on the empty assignment. Second, the first unassigned variable in π is assigned to its value in φ_{master}, followed by unit propagation of this value. A period ends when all variables are assigned a value in φ_{active}. Notice that *conflicts* - clauses with all literals assigned to false - are more or less neglected, depending on the implementation. A new period starts with the resulting φ_{active} as initial φ_{master} and a new ordering of the variables.

Example 4. Consider the example formula and initial settings below. Unassigned values in φ_{active} are denoted by $*$.

$$\begin{aligned}
\mathcal{F}_{\text{example}} &:= (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \\
&\quad\ (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \\
\varphi_{\text{master}} &:= \{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0\} \\
\varphi_{\text{active}} &:= \{x_1 = *, x_2 = *, x_3 = *, x_4 = *\} \\
\pi &:= (x_2, x_1, x_4, x_3)
\end{aligned}$$

Since the formula contains no unit clauses, the algorithm starts by selecting the first variable from the ordering - x_2. We assign this variable to true (as in φ_{master}) and perform unit propagation. Due to $\neg x_2 \vee \neg x_3$ this results in one unit clause $\neg x_3$. Propagation of this unit clause - assigning x_3 to false - results in

Algorithm 1. FLIP_UNITWALK(φ_{master})

```
1: for i in 1 to MAX_PERIODS do
2:     if φ_master satisfies F then
3:         break
4:     end if
5:     π := random ordering of the variables
6:     φ_active := ∅
7:     for j in 1 to n do
8:         while unit clause u ∈ φ_active ∘ F do
9:             φ_active[ VAR(u) ] := TRUTH(u)
10:        end while
11:        if π(j) not assigned in φ_active then
12:            φ_active[ π(j) ] := φ_master[ π(j) ]
13:        end if
14:    end for
15:    if φ_active = φ_master then
16:        random flip variable in φ_active
17:    end if
18:    φ_master := φ_active
19: end for
20: return φ_master
```

unit clauses x_4, and $\neg x_4$. Because two complementary unit clauses have been generated we found a conflict. However, the UNITWALK algorithm does not resolve this conflict.

Instead, it continues by selecting[1] one of them, say $\neg x_4$, and assign x_4 to false. After this assignment $\varphi_{\text{active}} \circ \mathcal{F}$ does not contain unit clauses anymore. We conclude this period by assigning x_1 to its value in φ_{master}. This results in the full assignment $\varphi_{\text{active}} = \{x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0\}$. Notice that the new assignment does not satisfy clause $\neg x_2 \vee x_3 \vee x_4$.

Now, consider the same example, this time using a 4-bit assignment to all the variables. The reader must keep in mind that by parallelizing the former, we try to satisfy clauses in each bit position! Hence, variables may be flipped in multiple bits, and "conflict" means a conflict in some bit position. For the latter we shall use the term *bit-conflict*. Further, we keep using the term "truth value" for its multi-valued analogue. Notice that in the initial settings below, the first bit in φ_{master} equals the 1-bit example and that the ordering is the same.

$$\begin{aligned} \varphi_{\text{master}} &:= \{x_1 = 0110, x_2 = 1100, x_3 = 1010, x_4 = 0110\} \\ \varphi_{\text{active}} &:= \{x_1 = ****, x_2 = ****, x_3 = ****, x_4 = ****\} \\ \pi &:= (x_2, x_1, x_4, x_3) \end{aligned}$$

[1] In [6] the authors suggest to select the truth value used in φ_{master}. However, this is not implemented in the latest version of the solver and we consider it as a choice.

Again, we start by assigning x_2 to its value in φ_{master} followed by unit propagation. This will result in two unit clauses :

$$\begin{aligned}(x_1 = **** \vee x_2 = \texttt{1100}) &\Rightarrow x_1 := **\texttt{11}\\ (\neg x_2 = \texttt{0011} \vee \neg x_3 = ****) &\Rightarrow x_3 := \texttt{00}**\end{aligned}$$

One of them is selected, say x_1 and assigned to its value, resulting in:

$$(\neg x_1 = **\texttt{00} \vee x_2 = \texttt{1100} \vee x_3 = \texttt{00}**) \Rightarrow x_3 := \texttt{0011}$$

Now we assign x_3 which triggers three clauses:

$$\begin{aligned}(\neg x_2 = \texttt{0011} \vee x_3 = \texttt{0011} \vee \neg x_4 = ****) &\Rightarrow x_4 := \texttt{00}**\\ (\neg x_2 = \texttt{0011} \vee x_3 = \texttt{0011} \vee x_4 = \texttt{00}**) &\Rightarrow \textbf{bit}-\textbf{conflict}\\ (\neg x_3 = \texttt{1100} \vee \neg x_4 = \texttt{11}**) &\Rightarrow x_4 := \texttt{0000}\end{aligned}$$

When unit propagation stops, only the first two bits of x_1 are still undefined. These bits are set to their value in φ_{master} assigning all variables. The period ends with $\varphi_{\text{active}} = \{x_1 = \texttt{0111}, x_2 = \texttt{1100}, x_3 = \texttt{0011}, x_4 = \texttt{0000}\}$ - which satisfies the formula in the third and fourth bit.

The reader may check that: (1) The order in which unit clauses are propagated, as well as the order in which clauses are evaluated is not fixed. The order influences φ_{active} in case of conflicts. For example, evaluating $\neg x_2 \vee x_3 \vee x_4$ before $\neg x_2 \vee x_3 \vee \neg x_4$ results in a different final φ_{active}. (2) In the 4-bit example the third and fourth bit are the same for all variables. This effect could reduce the parallelism, because the algorithm as such does not intervene here and in fact maintains this collapse. This effect is not restricted to formulas with few variables. During our experiments we frequently detected a convergence to identical assignments over a considerable number of bit positions (sometimes even over all 32 positions, when using a 32-bit processor). We implemented a fast detection algorithm which replaces a duplicate with a new random assignment. Due to page limitations we cannot go into detail at this stage. Notice however that by doing so the first "communication" aspect is introduced.

4 Implementation UnitMarch

4.1 Unit Propagation

The UnitPropagation procedure within the UnitWalk algorithm is not confluent: Different implementations yield different results. In short, two design decisions have to be made:

- In case of multiple unit clauses: which one to select for propagation;
- In case of a conflict: whether or how to act.

The most recent UnitWalk (version 1.003) implements the following UnitPropagation procedure: Unit clauses are stored in a multi-set (a set that can contain duplicate elements) data-structure. For each iteration a random element from the multi-set is selected. If the complement of the selected unit clause also occurs in the multi-set - meaning a conflict - all occurrences of x and $\neg x$ are removed from the multi-set. The algorithm continues with the next random element - see algorithm 2. Notice that this is a defensive flip strategy: Because of the removal, the truth value for x in φ_{active} tends to be the one copied from φ_{master}.

Algorithm 2. UnitPropagation_MultiSet ( )

```
 1: while UnitMultiSet is not empty do
 2:     x := random element from UnitMultiSet
 3:     remove all occurrences of x in UnitMultiSet
 4:     if unit clause ¬x also occurs in UnitMultiSet then
 5:         remove all occurrences of ¬x in UnitMultiSet
 6:     else
 7:         φ_active[ VAR(x) ] := TRUTH(x)
 8:         for all clauses C_i in which ¬x occurs do
 9:             if C_i becomes a unit clause then
10:                 add C_i to UnitMultiSet
11:             end if
12:         end for
13:     end if
14: end while
```

In our implementation we took a slightly different approach, since the above algorithm was hard to implement efficiently in a multi-bit version. Instead of the multi-set we used a queue (first in, first out) data-structure - see algorithm 3: Unit clauses are selected in the order they are added to the queue. In general, "early" generated unit clauses will have more bits assigned (at time of propagation) compared to "recent" unit clauses. Therefore the queue seems a useful data-structure since it always propagates the "earliest" unit clause left.

In addition, conflicts are handled differently: The queue is not allowed to contain complementary or duplicate unit clauses. The truth value of the first generated unit clause will be used during the further propagation. Notice that this flip strategy is more offensive: Given a bit-conflict, the truth value of the variable is flipped in approximately half of the cases. As we will see in the results (section 5), both implementations yield comparable results.

4.2 Detection of Unit Clauses

The UnitWalk algorithm spends most computational time in detecting which clauses became unit clauses given an expansion of φ_{active}. If a variable is assigned a Boolean value, all clauses in which it occurs with complementary polarity are potential unit clauses. In a 1-bit implementation, only one unit clause could be detected in such a potential clause, while in a multi-bit implementation multiple unit clauses could be detected:

Algorithm 3. UnitPropagation_Queue ( )

```
1: while UnitQueue is not empty do
2:     x := removed front element from UnitQueue
3:     for all clauses C_i in which ¬x occurs do
4:         if C_i becomes a unit clause then
5:             y := remaining literal in C_i
6:             φ_active[ VAR(y) ] := TRUTH(y)
7:             if y not in UnitQueue then append y to UnitQueue
8:         end if
9:     end for
10: end while
```

Example 6. Given $\varphi_{\text{active}} = \{x_1 = \texttt{010*}, x_2 = \texttt{10*1}, x_3 = \texttt{101*}, x_4 = \texttt{*001}\}$ with x_3 as unit clause to be propagated and potential clause $x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4$.

$$(x_1 = \texttt{010*} \vee \neg x_2 = \texttt{01*0} \vee \neg x_3 = \texttt{010*} \vee x_4 = \texttt{*001}) \;\Rightarrow\; x_2 := \texttt{1001},\; x_4 := \texttt{1001}$$

In general, all literals besides the propagation literal are potential unit clauses.

Encoding. Since each bit in φ_{active} consists of three possible values (`*`,`0`,`1`), we used two bits to encode each value: `00 = *`, `01 = 0`, `10 = 1`, and `11 =` **bit-conflict**.[2] We used an array φ^+_- in which both x_i and $\neg x_i$ have a separate assignment: The first bit of each value is stored in x_i while the second bit is stored in $\neg x_i$. For example:

$$\varphi_{\text{active}}[x] = \texttt{101**0*1} \text{ is stored as } \begin{cases} \varphi^+_-[\;\; x] = \texttt{10100001} \\ \varphi^+_-[\neg x] = \texttt{01000100} \end{cases}$$

Using φ^+_- we can compute the unit clauses as below. Conflicts are ignored by only allowing unassigned bits - computed by $\texttt{NOT}(\varphi^+_-[x_i]\ \texttt{OR}\ \varphi^+_-[\neg x_i])$ - to be assigned. Back to the example:

$$\begin{aligned} x_1 &:= \varphi^+_-[x_3]\ \texttt{AND NOT}(\varphi^+_-[x_1]\ \texttt{OR}\ \varphi^+_-[\neg x_1])\ \texttt{AND}\ \varphi^+_-[x_2]\ \texttt{AND}\ \varphi^+_-[\neg x_4] \\ \neg x_2 &:= \varphi^+_-[x_3]\ \texttt{AND}\ \varphi^+_-[\neg x_1]\ \texttt{AND NOT}(\varphi^+_-[x_2]\ \texttt{OR}\ \varphi^+_-[\neg x_2])\ \texttt{AND}\ \varphi^+_-[\neg x_4] \\ x_4 &:= \varphi^+_-[x_3]\ \texttt{AND}\ \varphi^+_-[\neg x_1]\ \texttt{AND}\ \varphi^+_-[x_2]\ \texttt{AND NOT}(\varphi^+_-[x_4]\ \texttt{OR}\ \varphi^+_-[\neg x_4]) \end{aligned}$$

The above shows a potential disadvantage of the multi-bit propagation: To check whether a clause of size k becomes a unit clause and to determine the remaining literal is not trivially computed in $\mathcal{O}(k)$ steps - as is the case with 1-bit propagation. However, a $\mathcal{O}(k)$ implementation can be realized by splitting the computation:

– Compute the *unit mask* - a multi-bit Boolean which is true on all positions with exactly one not falsified literal (denoted by $M_{\text{NF}=1}$);

[2] The bit-conflict value is not possible within our implementation.

– Use the unit mask to quickly determine the newly created unit clauses: All literals that are unassigned at a true position in the unit mask became unit.

To compute $M_{\mathrm{NF}=1}$, we use two auxiliary masks, $M_{\mathrm{NF}\geq 1}$ and $M_{\mathrm{NF}\geq 2}$. The masks denote multi-bit Booleans which are true on all positions with at least one (and two, respectively) falsified literals and false elsewhere. Notice that $M_{\mathrm{NF}=1}$:=$M_{\mathrm{NF}\geq 1}$ `XOR` $M_{\mathrm{NF}\geq 2}$. For each literal l_i in a clause we update $M_{\mathrm{NF}\geq 1}$ and $M_{\mathrm{NF}\geq 2}$ by the following two rules:

$$M_{\mathrm{NF}\geq 2} := (\ M_{\mathrm{NF}\geq 2}\ \texttt{OR}\ \texttt{NOT}(\varphi^+_-[\ \neg l_i\ ])\ )\ \texttt{AND}\ M_{\mathrm{NF}\geq 1}$$
$$M_{\mathrm{NF}\geq 1} := M_{\mathrm{NF}\geq 1}\ \texttt{OR}\ \texttt{NOT}(\varphi^+_-[\ \neg l_i\ ])$$

By negating the operations above, the computation becomes more efficient. Algorithm 4 shows the proposed implementation.

Algorithm 4. ComputeUnitMask (clause C_y)

```
1: M_I := ALL_BITS_TRUE, M_II := ALL_BITS_TRUE
2: for i in 1 to |C_y| do
3:     M_II := (M_II AND φ⁺_-[ ¬l_{y,i} ]) OR M_I
4:     M_I := M_I AND φ⁺_-[ ¬l_{y,i} ]
5: end for
6: return M_I XOR M_II
```

Once $M_{\mathrm{NF}=1}$ is computed ($M_{\mathrm{NF}=1}$ = `1010` in the example) we can determine the newly create unit clauses. For the example we only need the computations:

$$x_1 := M_{\mathrm{NF}=1}\ \texttt{AND}\ \texttt{NOT}(\varphi^+_-[x_1]\ \texttt{OR}\ \varphi^+_-[\neg x_1])$$
$$\neg x_2 := M_{\mathrm{NF}=1}\ \texttt{AND}\ \texttt{NOT}(\varphi^+_-[x_2]\ \texttt{OR}\ \varphi^+_-[\neg x_2])$$
$$x_4 := M_{\mathrm{NF}=1}\ \texttt{AND}\ \texttt{NOT}(\varphi^+_-[x_4]\ \texttt{OR}\ \varphi^+_-[\neg x_4])$$

5 Results

We implemented the UnitWalk algorithm as a multi-bit local search solver using UnitPropagation_Queue. The resulting solver, called UnitMarch, can be used for any number of bits. We added a method which replaces double assignments with new random assignments (see section 3). The performance of UnitMarch is compared with the latest version of UnitWalk[3].

The latter is a hybrid solver: If after a number of periods the number of unsatisfied clauses is not reduced the solver switches to WalkSat [8]. If that algorithm does not find a solution after a multitude of flips it switches back, etc. Because we wanted to compare the influence of multi-bit search on the pure UnitWalk algorithm, the switching was disabled.

[3] version 1.003 available from `http://logic.pdmi.ras.ru/~arist/UnitWalk/`

Table 1. Comparison between the performance - in average number of periods and average time and standard deviation - of UnitWalk, UnitMarch 1-bit, and UnitMarch 32-bit on various benchmarks. The presented data averages runs using 100 different random seeds.

	UnitWalk 1.003		UnitMarch 1-bit		UnitMarch 32-bit	
	periods	**time**	**periods**	**time**	**periods**	**time**
aim-2-1-1	119336	6.13 (6.36)	37520	1.62 (1.65)	1339	**0.32** (0.33)
aim-2-1-2	1395975	73.56 (71.97)	1001609	44.67 (43.37)	45934	**11.35** (10.68)
aim-2-1-3	26487	1.40 (1.39)	12147	0.53 (0.60)	646	**0.16** (0.15)
aim-2-1-4	57794	3.13 (3.01)	30708	1.38 (1.58)	945	**0.23** (0.22)
aim-3-4-1	89923	7.57 (7.05)	62191	3.19 (3.07)	2134	**1.40** (1.42)
aim-3-4-2	99744	8.43 (7.98)	181623	9.33 (8.51)	5838	**3.81** (3.33)
aim-3-4-3	51898	4.33 (4.07)	20870	1.7 (0.90)	738	**0.48** (0.45)
aim-3-4-4	264125	21.96 (17.79)	240856	21.21 (13.43)	6234	**4.29** (3.15)
bw-large.b	441	0.32 (0.33)	311	0.18 (0.13)	13	**0.05** (0.03)
bw-large.c	13870	47.61 (40.90)	9342	19.85 (22.05)	498	**7.63** (7.44)
dlx2-bug17	1102	6.40 (9.53)	432	2.31 (2.80)	7	**0.43** (0.41)
dlx2-bug39	2830	6.78 (6.13)	1899	4.38 (3.72)	69	**1.33** (1.76)
dlx2-bug40	1632	3.96 (4.02)	988	2.34 (2.20)	26	**0.55** (0.55)
flat200-05	19384	3.46 (3.40)	19880	2.19 (2.35)	704	**0.81** (0.75)
flat200-24	5247	0.98 (1.02)	5145	0.56 (0.56)	130	**0.16** (0.18)
flat200-39	12142	2.16 (2.29)	12048	1.31 (1.21)	391	**0.44** (0.45)
flat200-48	2941	0.52 (0.54)	2346	0.26 (0.25)	84	**0.10** (0.10)
flat200-64	6406	1.14 (1.03)	6799	0.75 (0.75)	268	**0.34** (0.35)
logistics-a	1970338	636.47 (563.21)	863165	369.09 (383.97)	25100	**55.97** (43.53)
logistics-b	6313	1.91 (2.24)	11878	5.43 (5.76)	354	**0.73** (0.63)
logistics-c	133572	72.16 (69.36)	310450	228.49 (224.92)	9803	**34.19** (31.75)
logistics-d	23	0.11 (0.07)	24	**0.08** (0.04)	5	0.11 (0.03)
par16-1	14245	4.97 (4.73)	11267	2.65 (2.85)	365	**0.21** (0.20)
par16-2	21417	7.43 (8.08)	20601	5.05 (5.18)	702	**0.42** (0.34)
par16-3	17913	6.31 (7.04)	16872	3.98 (3.93)	551	**0.33** (0.42)
par16-4	16955	5.94 (5.77)	14087	3.33 (3.47)	523	**0.34** (0.32)
par16-5	18889	6.60 (6.70)	23028	5.41 (5.00)	640	**0.36** (0.36)
qg1-08	101390	424.17 (399.59)	121127	362.74 (377.55)	4229	**127.57** (120.87)
qg2-08	803258	3404.49 (3501.46)	1005351	4360.92 (4518.23)	26223	**991.23** (967.20)
qg3-08	165	0.08 (0.06)	166	0.10 (0.10)	5	**0.03** (0.03)
qg4-09	1344	1.10 (0.96)	2098	1.82 (1.66)	66	**0.53** (0.53)
qg5-11	591	1.92 (1.82)	670	2.13 (2.00)	23	**0.82** (0.68)
qg7-13	92600	492.66 (465.71)	98172	408.35 (419.56)	2937	**171.63** (146.69)
uf250-054	307317	33.69 (35.84)	472970	30.03 (27.82)	14851	**10.74** (11.57)
uf250-062	42137	4.60 (4.85)	88670	5.61 (5.44)	2427	**1.74** (1.84)
uf250-071	135296	14.49 (12.79)	218375	13.92 (13.70)	6404	**4.59** (4.66)
uf250-072	126387	13.91 (13.33)	172789	10.95 (9.81)	5624	**4.10** (4.28)
uf250-093	92110	9.78 (9.71)	146132	9.23 (8.37)	4521	**3.25** (2.94)

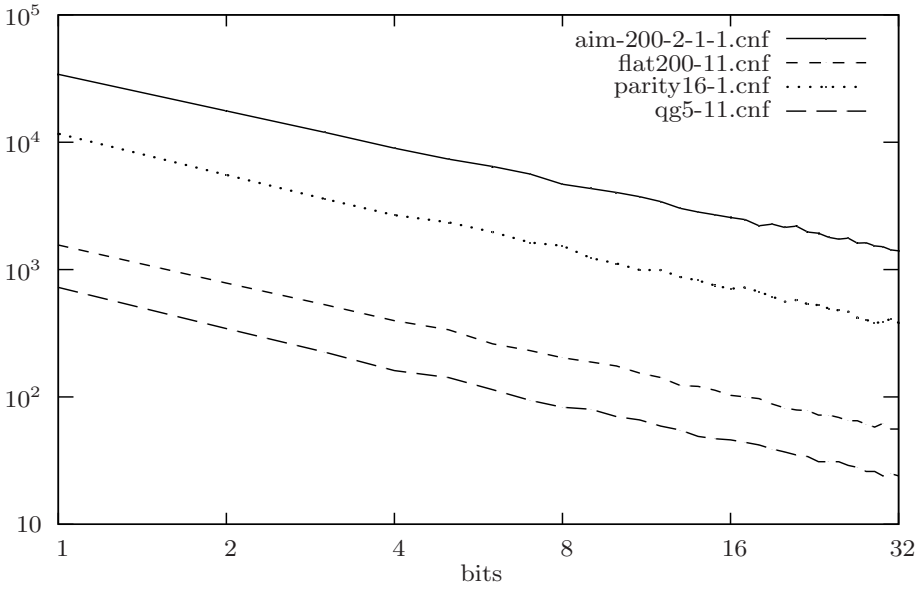


Fig. 1. Average number of periods by UnitMarch using different number of bits. Averages are computed using 1000 random seeds. Two logarithmic axes are used.

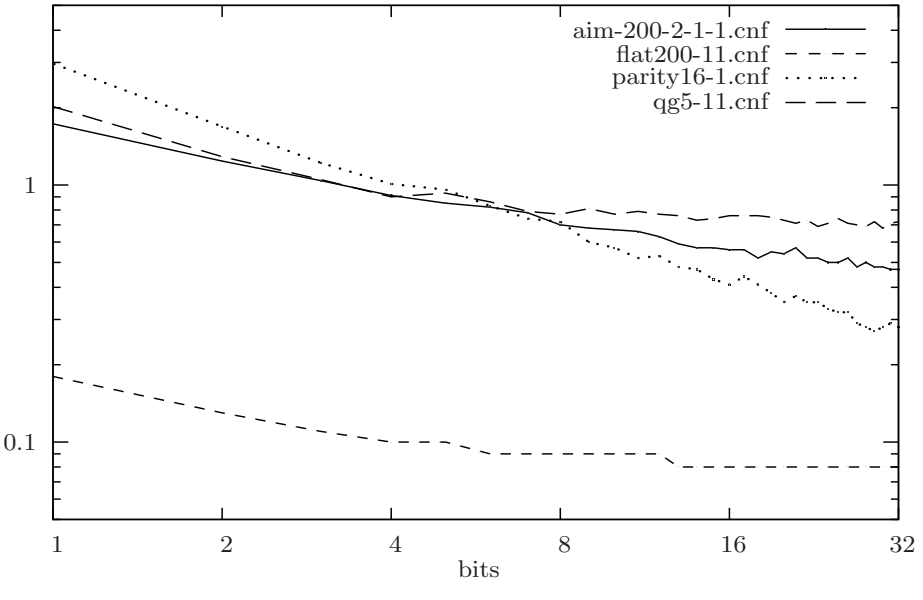


Fig. 2. Average time (in seconds) by UnitMarch using different number of bits. Averages are computed using 1000 random seeds. Two logarithmic axes are used.

Table 1 shows a comparison between UnitWalk, UnitMarch 1-bit and UnitMarch 32-bit on various benchmarks. Besides the *dlx2-bugXX* family,[4] all benchmarks can be found on SATlib[5] along with a description. For each solver, we set `MAX_PERIODS` $:= \infty$. We used 100 random seeds for all benchmarks.

The solvers UnitWalk and UnitMarch 1-bit show comparable performance. First, the number of periods executed per second is almost equal for all checked benchmarks. This shows that our implementation, with some overhead for parallelization, is fast enough on the benchmarks at hand. Second, the average number of periods between the two versions is comparable. Although they differ slightly between instances, no clear winner shows itself. Hence, the UnitPropagation_Queue procedure shows comparable to the UnitPropagation_MultiSet procedure in terms of performance.

Comparing the 1-bit solvers with UnitMarch 32-bit shows that the latter is the clear winner on almost all experimented instances. We found few exceptions (see *logistics-d*); all having less than 100 periods on the three solvers. Apparently, multi-bit search as implemented is not effective on these easy instances. Figures 1 and 2 present the effect of using different numbers of bits in more detail. Both figures use logarithmic axes - thus $f(x) = \frac{c}{x}$ is represented as a straight line. Four benchmarks are tested for all bits sizes 1 to 32. Using double logarithmic scaling, these instances show a linear dependency between the average number of periods and the number of used bits. The average time is also diminished on all these instances, although this reduction varies per instance. Notice that on all these instances the trend is strictly decreasing. It could be expected that computers with a k-bit architecture with $k > 32$ will boost performance even further.

6 Conclusions and Future Work

Our first observation is that propositional Boolean formulas with n variables can be mathematically elegantly checked on feasibility with a single assignment using the idempotents modulo the product of the first 2^n primes. Compared to conventional checking algorithms, the above just exchanges time for space. However, the architecture of today's computers is 32- or 64-bit - which enables execution of 32 (64) 1-bit operations simultaneously. Although many algorithms do not seem suitable for this kind of parallelism, the UnitWalk algorithm appears to be a good first candidate, as well as a state-of-the-art Sat solver [2].

Our multi-bit implementation of this algorithm, called UnitMarch, shows that this algorithm can be parallelized in such a way that the 1-bit version has comparable performance with the UnitWalk solver. Using double logarithmic scaling, these instances show a linear dependency between the average number of periods and the number of used bits. Most importantly, the average time to solve instances is largely reduced by using the 32-bit version.

[4] available from `http://www.miroslav-velev.com/sat_benchmarks.html`
[5] `http://www.satlib.org`

The implementations of UnitWalk and UnitMarch are currently comparable (regardless the multi-bit feature) but are far from optimal: For instance, in both solvers unit clauses in the original CNF are propagated in each period. Another performance boost is expected by adding (redundant) clauses - for instance as implemented in the local search solver $\mathsf{R}^{+}\mathsf{AdaptNovelty}^{+}$ [1] - because they will increase the number of unit propagations. Finally, further experiments (not presented in this paper) showed that by ordering the variables less randomly and more based on multi-bit heuristics results in improved performance on many benchmarks. Developing enhancements (like replacement of duplicate assignments) and effective multi-bit heuristics is under current research.

Acknowledgments

The authors would like to thank Denis de Leeuw Duarte for his contributions in the development of UnitMarch and Sean Weaver for his comments.

References

1. Anbulagan, Duc Nghia Pham, John K. Slaney, Abdul Sattar, *Old Resolution Meets Modern SLS*. AAAI-05 (2005), 354–359.
2. D. Le Berre and L. Simon, *The essentials of the SAT03 Competition*. Springer Verlag, Lecture Notes in Comput. Sci. **2919** (2004), 452–467.
3. W. Blochinger, C. Sinz, and W. Kuchlin, *Parallel propositional satisfiability checking with distributed dynamic learning*. Parallel Computing, **29**(7) (2003), 969-994.
4. M. Boehm and E. Speckenmeyer, *A fast parallel SAT-solver - efficient workload balancing*. Ann. Math. Artif. Intell. **17**(3-4) (2006), 381-400.
5. Frank M. Brown. *Boolean Reasoning: The Logic of Boolean Equations*.Kluwer Academic Publishers, Dordrecht (1990)
6. E. A. Hirsch and A. Kojevnikov, *UnitWalk: A new SAT solver that uses local search guided by unit clause elimination*. Ann. Math. Artif. Intell. **43**(1) (2005), 91–111.
7. F. Krohm, A. Kuehlmann, and A. Mets. *The Use of Random Simulation in Formal Verification*. Proc. of Int'l Conf. on Computer Design (1996), 371–376.
8. Bart Selman, Henry Kautz, and Bram Cohen. *Local search strategies for satisfiability testing*. In David S. Johnson and Michael A. Trick, editors, Cliques, Coloring, and Satisfiability: the Second DIMACS Implementation Challenge (1996), 521-532.
9. H. Zhang, M. P. Bonacina, and J. Hsiang. *PSATO: A distributed propositional prover and its application to quasigroup problems*. Journal of Symbolic Computation **21** (1996), 543-560.

Satisfiability with Exponential Families

Dominik Scheder and Philipp Zumstein

Institute of Theoretical Computer Science, ETH Zürich
8092 Zürich, Switzerland
dscheder@inf.ethz.ch, zuphilip@inf.ethz.ch

Abstract. Fix a set $S \subseteq \{0,1\}^*$ of exponential size, e.g. $|S \cap \{0,1\}^n| \in \Omega(\alpha^n), \alpha > 1$. The S-SAT problem asks whether a propositional formula F over variables $v_1, \ldots, v_n$ has a satisfying assignment $(v_1, \ldots, v_n) \in \{0,1\}^n \cap S$. Our interest is in determining the complexity of S-SAT. We prove that S-SAT is NP-complete for all context-free sets S. Furthermore, we show that if S-SAT is in P for some exponential S, then SAT and all problems in NP have polynomial circuits. This strongly indicates that satisfiability with exponential families is a hard problem. However, we also give an example of an exponential set S for which the S-SAT problem is not NP-hard, provided P $\neq$ NP.

Keywords: satisfiability, context-free grammars, VC-dimension, NP-hardness, polynomial circuits.

1 Introduction

Given a set $S \subseteq \{0,1\}^*$ of all assignments, the S-SAT problem asks whether for a formula F over n variables there is an assignment $x \in S_n := S \cap \{0,1\}^n$ that satisfies F (F is then called S-satisfiable). The other assignments $\{0,1\}^n \setminus S$ can be seen as assignments which are a priori forbidden. If $|S_n|$ is polynomial in n and S_n can be enumerated in polynomial time then S-SAT is in P. To exclude this case we concentrate on *exponential families*, which are defined next.

Definition 1.1. *A monotonically increasing sequence $Q = (n_j)_{j \in \mathbb{N}} \subseteq \mathbb{N}$ has* polynomial gaps *if there is a polynomial $p(n)$ such that*

$$n_{j+1} \leq p(n_j)$$

for all $j \in \mathbb{N}$.

For example, define $n_j = 2^j$. Then $n_{j+1} = 2n_j$, so $p(n) := 2n$ shows that this sequence has polynomial gaps. This means, a sequence (n_j) can increase exponentially in j and still have polynomial gaps. Note that we can always assume w.l.o.g. that $p(n)$ is strictly increasing.

Definition 1.2. *The family $(S_n)_{n \geq 0}$ is called* exponential *if there exists $\alpha > 1$ and a sequence Q with polynomial gaps such that*

$$\forall n \in Q : |S_n| \geq \alpha^n .$$

We also say $S = \bigcup_{n \geq 1} S_n$ has exponential size.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 148–158, 2007.

For example families with $|S_n| \in \Omega(\alpha^n)$ are exponential (but we additionally allow to have some "gaps"). There are some subtleties involved with the definition of S-SAT. First, observe that we can interpret $x \in \{0,1\}^*$ as a truth assignment only if V, the set of variables, is *ordered*. Second, we require that V is given explicitly as a part of the input together with the formula F. To see why this is necessary, define $S_n = \{x \in \{0,1\}^n \mid x_n = 0\}$. Then the formula $v_1 \wedge v_2 \wedge v_3$ with $V = (v_1, v_2, v_3)$ is not S-satisfiable, but with $V = (v_1, v_2, v_3, v_4)$ it is. Note that we do not require every variable in V to occur in F (this does not affect our results but turns out to be a useful convention). For simplicity of notation, we agree that the variables of V are named $v_1, \ldots, v_n$, in this order. Finally we want to point out that S is some fixed language and therefore it is not part of the input.

The question whether S-SAT is NP-hard for all exponential S was first stated by Cooper [1] on his web page, though we are working with a more general notion of *exponential*. As far as we know, there have not been any further considerations about S-SAT neither by Cooper nor by anybody else.

Our Results

We prove that S-SAT is NP-complete for all exponential S that are context-free (Section 4). Further, we show that if S-SAT is in P for some exponential S, then SAT, and thus every problem in NP, has polynomial circuits (Section 5). This would imply that the polynomial hierarchy collapses to its second level [2]. Since this is widely believed to be false, it is a strong indication that S-SAT is a hard problem in general. However, we construct an exponential S such that S-SAT is not NP-hard, provided $\mathsf{P} \neq \mathsf{NP}$ (Section 6).

2 Some Observations

It is easy to show NP-hardness of S-SAT for $S_n = \{x \in \{0,1\}^n \mid x_1 = 0\}$ (and similar simple families): Let the formula F be an instance of SAT. We construct the formula F' which is identical to F but with every occurrence of x_1 replaced by $\bar{x}_1$ and vice versa. The formula F is satisfiable iff the formula $F \vee F'$ is S-satisfiable. This is a polynomial reduction from SAT to S-SAT.

If we view S itself as a language over the alphabet $\{0,1\}$, and therefore as a decision problem, we get the following connection:

Proposition 2.1. *S can be reduced to S-SAT in polynomial time.*

Proof. Given some $x = (x_1, \ldots, x_n) \in \{0,1\}^n$. Write $v^1 := v$ and $v^0 := \bar{v}$, respectively. Then x is the unique assignment in $\{0,1\}^n$ that satisfies the formula $F_x := v_1^{x_1} \wedge v_2^{x_2} \wedge \cdots \wedge v_n^{x_n}$ over V with $|V| = n$. Hence, F_x is S-satisfiable if and only if $x \in S_n$. Clearly, this is a polynomial reduction from S to S-SAT. □

Hence, S-SAT can have arbitrarily high complexity; it may even be undecidable. The next proposition demonstrates how we can employ a fast S-SAT algorithm, if existent, to solve SAT in significantly less than 2^n steps. We write $O^*(f(n))$ if we neglect polynomial factors.

Proposition 2.2. *Suppose there is some S with $|S_n| \geq \alpha^n$ for $\alpha > 1$ and all sufficiently large n. If S-SAT can be decided in time $O^*(\beta^n)$, then there is a randomized Monte Carlo algorithm for* SAT *with running time $O^*((2\beta/\alpha)^n)$.*

Proof. Let F be a satisfiable formula over a set V of variables, and let x be a satisfying assignment. For each variable $v \in V$, *switch* v with probability $1/2$, i.e. invert all its occurrences in F and its value according to the assignment x, resulting in a new formula F' and a new assignment x'. The assignment x' satisfies F' if and only if x satisfies the original formula F. Moreover, x' is uniformly distributed over $\{0,1\}^n$. Therefore, with probability $p := \Pr[x' \in S_n] \geq (\alpha/2)^n$ the formula F' is S-satisfiable. This can be tested in time $O^*(\beta^n)$. After repeating this process $(2/\alpha)^n$ times, the probability that at least one of the randomly generated formulas is S-satisfiable, is at least $1 - 1/e$, hence constant. On the other hand, if F is unsatisfiable, it will not become satisfiable by switching. We therefore have a Monte Carlo algorithm with running time $(2/\alpha)^n O^*(\beta^n)$. □

There are no known algorithms for SAT running in time $O^*(\gamma^n)$ for $\gamma < 2$, not even randomized ones. Proposition 2.2 with $\beta < \alpha$, therefore, is a first indication that S-SAT is a difficult problem.

In fact, the currently best known deterministic algorithm for 3-SAT (satisfiability of formulas in *conjunctive normal form* where every disjunction consists of at most 3 literals) can be viewed as a derandomized version of the randomized algorithm in the proof of Proposition 2.2: Let the *Hamming distance* $d(x, y)$ of two vectors $x, y \in \{0,1\}^n$ be the number of bits in which they differ. The *Hamming Ball* of radius r around x is, in analogy to the usual definition of a ball, the set $B_r(x) := \{y \in \{0,1\}^n \mid d(x,y) \leq r\}$. We look at the family $S_n = B_{\rho n}(\mathbf{0})$ where ρ is some constant. Then

$$|S_n| = \sum_{i=0}^{\rho n} \binom{n}{i} \approx 2^{H(\rho)n}, \quad H(t) = -t \log t - (1-t)\log(1-t).$$

Therefore $S = (S_n)_{n \geq 0}$ is an exponential family. For 3-CNFs, S-SAT can be decided in $3^{\rho n}$ steps (by splitting on 3-clauses), which for appropriately chosen ρ is much smaller than $2^{H(\rho)n}$. By choosing many Hamming balls centered at different points (randomly) and by choosing the optimal value of ρ this yields an algorithm deciding 3-SAT in $O^*(1.5^n)$ steps. Note that choosing a random point as center of the Hamming ball is equivalent to switching the formula randomly and keeping the Hamming ball centered at $(0, \ldots, 0)$ all time. It takes some additional effort to derandomize the algorithm. For details, see Dantsin et al. [3].

3 *S*-SAT and the VC-Dimension

To obtain a systematical way of proving NP-hardness of S-SAT (if possible), we exploit the notion of *shattering* and the *Vapnik-Chervonenkis-dimension* $d_{\mathrm{VC}}(S_n)$, short VC-dimension, of a set $S_n \subseteq \{0,1\}^n$. These concepts were first

introduced by Vapnik and Chervonenkis [4]. Let V with $|V| = n$ be an ordered set of variables. We say $I \subseteq [n]$ is *shattered* by S_n if any assignment to $V_I := \{v_i \mid i \in I\}$ can be realized by S_n. Formally, for every $x \in \{0,1\}^{|I|}$ there is a $y \in S_n$ with $y|_I = x$, where $y|_I$ denotes the $|I|$-bit vector $(y_i)_{i \in I}$. The VC-dimension d_{VC} is the size of a largest shattered set. Obviously, $0 \leq d_{\mathrm{VC}}(S_n) \leq n$. The intuition is that large sets have large VC-dimensions. This is quantified by the following lemma, which was proven several times independently, e. g. by Sauer [5].

Lemma 3.1. *Suppose* $d_{\mathrm{VC}}(S_n) \leq d \leq n/2$. *Then*

$$|S_n| \leq \sum_{i=0}^{d} \binom{n}{i} \leq 2^{H(\frac{d}{n})n}$$

where $H(x) = -x \log(x) - (1-x)\log(1-x)$ *is the binary entropy function.*

Corollary 3.2. *Suppose* $S \subseteq \{0,1\}^*$ *has exponential size. Then there is a polynomial* $q(n)$ *such that for each* $n \in \mathbb{N}$ *there exists* $N \leq q(n)$ *and an index set* $I \subseteq [N]$ *with* $|I| \geq n$ *such that* I *is shattered by* S_N.

Proof. Let $(n_j)_{j \in \mathbb{N}}$ be the sequence with polynomial gaps corresponding to the exponential family S, i.e. there is an $\alpha > 1$ and a polynomial $p(n)$ such that $n_{j+1} \leq p(n_j)$ and $|S_{n_j}| \geq \alpha^{n_j}$ for all j. Choose $\delta \in (0, 1/2]$ such that $H(\delta) = \log \alpha$ and k such that $n_k \leq \frac{n}{\delta} \leq n_{k+1} =: N$. By Lemma 3.1, $d_{\mathrm{VC}}(S_N) \geq \delta N \geq n$, so there exists a shattered set $I \subseteq [N]$ with $|I| \geq n$. Note that $N = n_{k+1} \leq p(n_k) \leq p(n/\delta) =: q(n)$, as required. □

Although we know that a large shattered set exists, it is not clear how we can compute it efficiently. Let us for the moment assume that we can. Then there is a polynomial reduction from SAT to S-SAT:

Theorem 3.3. *Let* $S \subseteq \{0,1\}^*$ *be of exponential size and let* $p(n)$ *be a polynomial. Suppose that for all* n, *we can compute, in time polynomial in* n, *some number* $N \leq p(n)$ *and some index set* $I \subseteq [N]$ *with* $|I| \geq n$ *that is shattered by* S_N. *Then* S-SAT *is* NP*-hard.*

Proof. The existence of such a I is guaranteed by Corollary 3.2. Suppose it can be computed efficiently. Let F be a formula over the variables $V_n = \{v_1, \ldots, v_n\}$. We construct a new formula F' over V_N by renaming each v_j occurring in F into v_{i_j} where $I \supseteq \{i_1, \ldots, i_n\}$. We claim that F is satisfiable iff F' is S-satisfiable. Suppose $x \in \{0,1\}^n$ satisfies F. Clearly, there is some $x' \in \{0,1\}^N$ satisfying F', since F and F' differ only in the names of their variables. Every assignment $y \in \{0,1\}^N$ that agrees with x' in the variables $(v_{i_1}, \ldots, v_{i_n})$ also satisfies F'. It follows from the definition of shattering that S_N contains such a y. Hence, F' is S-satisfiable. The reverse direction is clear. This polynomial reduction shows that S-SAT is NP-hard, under these conditions. □

Why does this method not work general? The difficulty is that we do not know which subset of variables is shattered, we only know that there is one. It is also futile to try to compute a large shattered set directly from S_n, since a polynomial reduction cannot deal with S_n explicitly, as $|S_n|$ is exponential in n (at least the S_n we are interested in is). Note that the brute force approach to computing the VC-dimension of a set will take time *polynomial* in $|S_n|$, if $|S_n|$ is exponentially in n. This is in contrast to the result of Papadimitriou and Yannakakis [6] that computing the VC-dimension of an explicitly given S_n (of size not necessarily exponential in n) is LOGNP-complete, hence unlikely to be in P. But this is no help to us: though computing the VC dimension takes time polynomial in $|S_n|$, observe that $|S_n|$ is itself exponential in n.

We see that a polynomial reduction from SAT to S-SAT must somehow have certain implicit knowledge of $S = \cup S_n$. One way would be a (regular, context-free, ...) grammar of S (if there is one).

Theorem 3.4. *If $S \subseteq \{0,1\}^*$ is a regular language and $S_n := S \cap \{0,1\}^n$, then $d_{\mathrm{VC}}(S_n)$ and a shattered set $I \subseteq [n]$ of this size can be computed in $O(n^2)$ (where the hidden constant factor is doubly exponential in the size of the regular grammar).*

The proof of this theorem is quite technical and is therefore omitted here. Instead, we will prove a similar theorem for context-free languages where we do not insist on computing a *largest* shattered index set, but only a *sufficiently large* one.

4 NP-Completeness of Context-Free S-SAT

In this section, we prove that S-SAT is NP-complete if S is a context-free language and has exponential size. It suffices to show how to find a large shattered index set. To be more precise, for any given n, we will find some $N \in O(n)$ and $I \subseteq [N]$ with $|I| \geq n$ such that I is shattered by S_N. In combination with the results from Section 3, this proves NP-hardness. It is clear that S-SAT is in NP if S is context-free, since deciding whether $x \in S$ and verifying that x is satisfying can be done in polynomial time.

In the following, we denote the nonterminal symbols appearing in the context-free grammar for S by upper case letters $S_0, A, B, C, \ldots$, where S_0 is the starting symbol. The only terminal symbols are $0, 1$. All rules in a context-free grammar are of the form $A \vdash w$ for a word w possibly containing nonterminals. $A \vdash^* w$ means that w can be derived from A in finitely many steps. Finally, the length of a word x is denoted by $|x|$.

Let S be a context-free, exponential language which is generated by the grammar G. All calculations on the grammar can be done in advance and therefore do not contribute to the running time. In particular, we may assume that G does not contain *useless* nor *unreachable* nonterminal symbols, i.e. for every nonterminal A, we have $A \vdash^* x$ for some $x \in \{0,1\}^*$, and $S \vdash^* w$ for some w with

$A \in w$. We call such a grammar *reduced*. For a nonterminal A, define

$$\ell(A) := \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^* : A \vdash^* xAy\} ,$$
$$r(A) := \{y \in \{0,1\}^* \mid \exists x \in \{0,1\}^* : A \vdash^* xAy\} .$$

Call some $X \subseteq \{0,1\}^*$ *commutative* if $xy = yx$ for all $x, y \in X$.

Lemma 4.1 (Ginsburg [7], Theorem 5.5.1). *Let G be a reduced context-free grammar and let $L(G)$ be the language generated by G. Then $|L(G) \cap \{0,1\}^n|$ is polynomial in n if and only if for every nonterminal A, $\ell(A)$ and $r(A)$ are commutative.*

Theorem 4.2. *Suppose $S \subseteq \{0,1\}^*$ has exponential size and is a context-free language. Then S-SAT is* NP*-complete.*

Proof. We will show how to compute large shattered sets, for every n. Let G be a reduced context-free grammar for S. Since S has exponential size, $|S_n|$ is surely not polynomial in n. Therefore, Lemma 4.1 implies that there is a nonterminal A such that $\ell(A)$ or $r(A)$ is not commutative. Suppose w.l.o.g. that $\ell(A)$ is not commutative, and let $x_1, x_2 \in \ell(A)$ such that $x_1x_2 \neq x_2x_1$. Hence, there is a position i such that w.l.o.g. $(x_1x_2)_i = 0$ and $(x_2x_1)_i = 1$. By definition, there are $y_1, y_2 \in \{0,1\}^*$ such that $A \vdash^* x_1Ay_1$ and $A \vdash^* x_2Ay_2$. By applying k times either $A \vdash^* x_1x_2Ay_2y_1$ or $A \vdash^* x_2x_1Ay_1y_2$, we can create arbitrary 0s and 1s at the positions $i + k \cdot |x_1x_2|$ for any k. In order to reach A from S_0, we use $S_0 \vdash^* aAb$, and in the end we use $A \vdash^* w$ to obtain a word in $\{0,1\}^*$ for some $a, b, w \subseteq \{0,1\}^*$. Hence if we set $N := |a| + |b| + |w| + n(|x_1x_2| + |y_1y_2|)$, then $I := \{|a| + k|x_1x_2| + i \,:\, 0 \leq k \leq n-1\}$ is of size n, and it is shattered by S_N. All these calculations can be done in time $O(n)$ and N is linear in n. Thus, by Theorem 3.3, S-SAT is NP-hard. Since S-SAT $\in$ NP, it is NP-complete. □

5 *S*-SAT and Polynomial Circuits

In the previous section, we have seen that if we can efficiently compute large shattered sets, then S-SAT is NP-hard. If we cannot compute those sets, then we do not have a systematic way of proving NP-hardness (although there are simple examples where large shattered sets of S_n cannot be computed at all, and still S-SAT is NP-hard). However, we will prove a result that is "almost as good" as proving NP-completeness: if S-SAT is in P for some exponential S, then SAT has polynomial circuits.

Since boolean circuits are standard terminology in complexity theory, we do not give a formal definition. Furthermore, because we are interested in the *size* of a circuit, i.e. the number of its gates, and not in the *depth*, it does make a difference whether we allow bounded fan-in or not. For an overview of boolean circuits in complexity theory, see [8].

Definition 5.1. *A* circuit family *is a sequence $\mathcal{C} = (C_1, C_2, \dots)$ of boolean circuits, where each C_n has n input gates. If each C_n has exactly one output gate,*

then $\mathcal{C}$ computes a function $f : \{0,1\}^* \to \{0,1\}$, or equivalently, decides a language $L \subseteq \{0,1\}^*$.

If the size of C_n grows polynomially in n, then $\mathcal{C}$ is a polynomial circuit family*. If there exists an algorithm that computes and outputs C_n in time polynomial in n, we call $\mathcal{C}$ a* uniform polynomial circuit family*.*

It is not hard to show that a language $L \in \{0,1\}^*$ can be decided by uniform polynomial circuits if and only if it is in P. There are even undecidable languages with (nonuniform, of course) polynomial circuits. However, there are good reasons to believe that NP-complete problems do not have polynomial circuits, whether uniform or not: Karp and Lipton [2] showed that if NP-complete problems have polynomial circuits, then the polynomial hierarchy collapses to its second level. The connection to S-SAT is immediate:

Theorem 5.2. *If S-*SAT *is in P for some exponential S, then* SAT *has (possibly nonuniform) polynomial circuits.*

Proof. From Corollary 3.2, we know that for each n there exists an $N \leq q(n)$ and an index set $I \subseteq [N]$ with $|I| \geq n$ such that I is shattered by S_N. For each n, there is a boolean circuit of polynomial size that takes a formula F over n variables as input and outputs a formula F' over N variables, where F' is identical to F, but with the all variables from F replaced by variables in I. Note that the circuit *exists*, though it might not be constructible in polynomial time. By assumption, there is a second circuit of polynomial size deciding S-SAT for formulas with N variables. This circuit can be constructed in polynomial time. Combining these two circuits yields a polynomial circuit deciding SAT. □

It might be possible that NP has polynomial circuits and still $\mathsf{P} \neq \mathsf{NP}$. Hence, this result is weaker than proving NP-hardness for S-SAT in general.

6 Some *S*-SAT Which Is Not NP-hard

In this section we will prove—under reasonable assumptions—that there is an exponential S such that S-SAT is not NP-hard. We will use a classical tool of complexity theory: diagonalization. Let us first introduce some notation. As we stated in Section 1, we assume that an instance of S-SAT always comes with an explicitly given set of variables $V = \{v_1, \ldots, v_n\}$. For a formula F, let $n(F)$ denote the size of this variable set, *not* the number of variables actually present in F. These sets can differ, as we have seen.

Definition 6.1. *A function φ mapping formulas to formulas is called a* SAT-reduction *if, for all satisfiable formulas F and unsatisfiable formulas F', we have $\varphi(F) \neq \varphi(F')$. If there exists an algorithm which computes φ in polynomial time, then we say that it is a* polynomial SAT-reduction*.*

Consider for example the mapping φ which maps every satisfiable formula to 1 and every unsatisfiable formula to 0. This φ is a SAT-reduction but it is not

polynomial (provided NP≠ P). It should be clear that any function φ, that does not fulfill the condition of being a SAT-reduction, is disqualified from being a reduction from SAT to any S-SAT in the first place.

Theorem 6.2. *Provided that* $\mathsf{P} \neq \mathsf{NP}$*, there is an* S *with* $|S_n| = 2^n$ *for at least every second* n*, and* SAT $\not\leq_p$ S-SAT*. Thus,* S *has exponential size and* S-SAT *is not* NP*-hard.*

Provided that $\mathsf{P} \neq \mathsf{NP}$, we will show that there are arbitrarily large formulas having preimages that are satisfiable for every polynomial SAT-reduction φ. Nnote that F might have several preimages, but according to the definition of a SAT-reduction, they are either all satisfiable or all unsatisfiable. If such a formula F has n variables, and G is one of its satisfiable preimages, then setting $S_n = \emptyset$ prevents φ from being a polynomial reduction from SAT to S-SAT, since G is satisfiable but $F = \varphi(G)$ is not S-satisfiable. We then choose such n_i for each polynomial SAT-reduction φ_i and set $S_n = \{0,1\}^n$ for all other remaining values of n. By leaving gaps between the n_i, we guarantee that S has exponential size.

Lemma 6.3. *Provided that* $\mathsf{P} \neq \mathsf{NP}$*, then for every polynomial* SAT*-reduction* φ*, there are arbitrarily large formulas (in terms of* $n(F)$*) with satisfiable preimages.*

Proof. For the sake of contradiction, suppose that there is some SAT-reduction φ and some n_0 such that $n(\varphi(F)) \leq n_0$ for all satisfiable F. Consider

$$\mathcal{F}_0 := \{\varphi(F) \mid F \text{ is a satisfiable formula}\}$$

the image of all satisfiable formulas. By assumption, all formulas in $\mathcal{F}_0$ have no more than n_0 variables, implying that $\mathcal{F}_0$ is finite. Clearly, F is satisfiable iff $\varphi(F) \in \mathcal{F}_0$. Thus, φ reduces SAT to the finite language $\mathcal{F}_0$. Since every finite language is in P, SAT is in P, too. This contradicts our assumption. □

Proof (of Theorem 6.2). The Lemma gives us functions $n(\varphi, n_0)$, $F(\varphi, n_0)$ such that $n(\varphi, n_0) \geq n_0$, and $F(\varphi, n_0)$ has exactly $n(\varphi, n_0)$ variables and has satisfiable preimages.

Let $\varphi_1, \varphi_2, \ldots$ be an enumeration of all polynomial SAT-reductions (there are countably many) and define

$$\begin{aligned} n_1 &:= n(\varphi_1, 0)\,, \\ n_{i+1} &:= n(\varphi_{i+1}, n_i + 2)\,. \end{aligned}$$

$$S_n := \begin{cases} \emptyset & \text{if } n = n_i \text{ for some } i; \\ \{0,1\}^n & \text{otherwise.} \end{cases}$$

First, note that $n_{i+1} - n_i \geq 2$. Therefore, if $S_n = \emptyset$, then $|S_{n-1}| = 2^{n-1}$. Hence at least half of the levels are "full". Second, suppose some φ_i reduces SAT to S-SAT. By construction, there is a satisfiable formula F such that $\varphi_i(F)$ has exactly n_i variables. Unfortunately, S_{n_i} is empty, so $\varphi_i(F)$ is not S-satisfiable, hence φ_i is not a reduction, which is a contradiction. Since every SAT-reduction appears as some φ_i in our sequence, the proof is complete. □

This is nice, but has the drawback that S might have gaps, i.e. not every level has exponential size. The next construction gives us an S that overcomes this deficiency.

Theorem 6.4. *Provided that* $\mathsf{RP} \neq \mathsf{NP}$*, there is an* S *with* $|S_n| \geq 2^{n-1}$ *for all* n *such that* S-SAT *is not* NP*-hard.*

The problem above was that, in order to ensure that for the satisfiable formula $F = F(\varphi, n_0)$, $\varphi(F)$ is not S-satisfiable, we had to set $S_n = \emptyset$ for $n = n(\varphi, n_0)$, creating a "gap" in S. Alternatively, we could set $S_n := \{0,1\}^n \backslash \mathrm{sat}(\varphi(F))$, where $\mathrm{sat}(\varphi(F))$ is the set of all assignment which satisfy $\varphi(F)$. Clearly this suffices to ensure that $\varphi(F)$ is not S-satisfiable, preventing φ from being a reduction from SAT to S-SAT. If, in addition, $\mathrm{sat}(\varphi(F))$ is small, $|S_n|$ will be exponential in n. Let us now first focus on what happens when it is never small.

Definition 6.5. *A* SAT*-reduction* φ *is referred to* sharp*, if there is some* n_0 *such that for all* F *with* $n := n(\varphi(F)) \geq n_0$*, the following two statements hold:*

(i) F *and* $\varphi(F)$ *are* SAT*-equivalent, that is, either both are satisfiable, or both are not*
(ii) if $\varphi(F)$ *is satisfiable, then* $|\mathrm{sat}(\varphi(F))| > 2^{n-1}$

The choice of 2^{n-1} is arbitrary. Any number x with $x/2^n > \epsilon > 0$ and $2^n - x$ being exponential would be good as well. The image of a sharp reduction consists of formulas with at most n_0 variables, unsatisfiable formulas, and formulas with a huge number of satisfying assignments.

Lemma 6.6. *If there is a polynomial sharp* SAT*-reduction* φ*, then* $\mathsf{RP} = \mathsf{NP}$.

Proof. We give a randomized algorithm for SAT with a bounded error probability. Similar to the proof of Lemma 6.3, define

$$\mathcal{F}_0 := \{\varphi(F) \mid F \text{ is satisfiable and } n(\varphi(F)) \leq n_0\}$$

Again, this set is finite. We compute satisfiability of some input formula F with $n(F) = n$ as follows: if $n(\varphi(F)) \leq n_0$, we simply check whether $\varphi(F) \in \mathcal{F}_0$, which can be done in constant time. Otherwise, either both F and $\varphi(F)$ are unsatisfiable, or both are satisfiable, but then $\mathrm{sat}(\varphi(F))$ is huge. Let x be a uniformly at random chosen assignment out of $\{0,1\}^n$ for $n = n(\varphi(F))$ and return *satisfiable* if x satisfies $\varphi(F)$ and *unsatisfiable* otherwise. If F is unsatisfiable, the algorithm always answers correctly, otherwise the answer is wrong with a probability $p \leq 1/2$. Thus SAT is in RP, and hence $\mathsf{RP} = \mathsf{NP}$. □

The contrapositive of Lemma 6.6 reads as follows: Provided that $\mathsf{RP} \neq \mathsf{NP}$, no polynomial SAT-reduction φ is sharp, which means that for all φ, n_0, there exist $n = n(\varphi, n_0) \geq n_0$, $F = F(\varphi, n_0)$, such that $\varphi(F)$ has n variables and one of the following holds:

(i) F and $\varphi(F)$ are not SAT-equivalent
(ii) they are SAT-equivalent, $\varphi(F)$ is satisfiable, and $|\mathrm{sat}(\varphi(F))| \leq 2^{n-1}$

Proof (of Theorem 6.4). Using the function $n(\varphi, n_0)$ and our sequence $\varphi_1, \varphi_2, \ldots$ of polynomial SAT-reductions, we define

$$\begin{aligned} n_1 &:= n(\varphi_1, 0)\,, & F_1 &:= F(\varphi_1, 0)\,, \\ n_{i+1} &:= n(\varphi_{i+1}, n_i + 1)\,, & F_{i+1} &:= F(\varphi_{i+1}, n_i + 1)\,. \end{aligned}$$

So the F_i are the formulas with n_i variables provided by the contrapositive of Lemma 6.6, and the n_i are all distinct. If case (i) above applies to F_i, we say n_i is of type (i), if case (ii) applies, n_i is of type (ii). We define S by

$$S_n := \begin{cases} \{0,1\}^n \setminus \mathrm{sat}(\varphi_i(F_i)) & \text{if } \mathrm{n} = \mathrm{n_i} \text{ is of type (ii);} \\ \{0,1\}^n & \text{otherwise.} \end{cases}$$

We claim that every φ fails to be a reduction from SAT to S-SAT. Take any φ_i. If n_i is of type (i), then F_i and $\varphi_i(F_i)$ are not SAT-equivalent, and since $S_{n_i} = \{0,1\}^{n_i}$, $\varphi_i(F_i)$ is S-satisfiable iff F_i is not satisfiable. Thus, φ is not a reduction from SAT to S-SAT. If n_i is of type (ii), then F_i and $\varphi_i(F_i)$ are both satisfiable, but $\varphi_i(F_i)$ is not S-satisfiable, since $S_{n_i} = \{0,1\}^{n_i} \setminus \mathrm{sat}(\varphi_i(F_i))$. Hence φ_i fails also in this case. Finally, note that $|S_n| \geq 2^{n-1}$ for all n. □

As one referee pointed out, Theorem 6.2 looks like a weaker version of Ladner's theorem [9], which states that there are *intermediate* languages $L \in \mathsf{NP} \setminus \mathsf{P}$ which are not NP-complete, provided that $\mathsf{P} \neq \mathsf{NP}$. In fact, we could use Ladner's theorem to define a set $S \subseteq \{0,1\}^*$ such that S-SAT is an intermediate language. Unfortunately, it is not clear whether such S is exponential according to Definition 1.2. Certainly, it is much less "dense" than the languages S defined in the proofs of Theorem 6.2 and Theorem 6.4, for which it holds that for all n $|\bigcup_{i \leq n} S_i| \in \Omega(2^n)$ and $|S_n| \geq 2^{n-1}$, respectively.

7 Conclusion

Let us go back to where we started. We were interested in the complexity of S-SAT, for some given $S \subseteq \{0,1\}^*$. We can restate the question:

> **Problem:** Find a large natural class $\mathcal{S} \subseteq 2^{\{0,1\}^*}$ of sets of assignments, such that S-SAT is NP-hard (or even NP-complete) for all $S \in \mathcal{S}$.

As we have seen, the set of all exponential context-free languages is such a class, while the class of all exponential languages is not such a class. Might it be that S-SAT is NP-complete for all exponential S in P? In the light of Ladner's theorem [9], this seems unlikely.

References

1. Cooper, J.: Josh Cooper's Math Pages: Combinatorial problems I like `http://www.math.sc.edu/~cooper/combprob.html`.
2. Karp, R., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: Enseign. Math. 28. (1982) 191–201
3. Dantsin, E., Goerdt, A., E. A.H., Kannan, R., Kleinberg, J., Papadimitriou, C., Raghavan, O., Schöning, U.: A deterministic $(2 - 2/(k + 1))^n$ algorithm for k-SAT based on local search. In: Theoretical Computer Science 289. (2002) 69–83
4. Vapnik, V., Chervonenkis, A.: On the uniform convergence of relative frequencies of events to their probabilities. Theory Prob. Appl. **16** (1971) 264–280
5. Sauer, N.: On the density of families of sets. In: J. Comb. Theory, Ser. (A). Volume 13. (1973) 145–147
6. Papadimitriou, C.H., Yannakakis, M.: On limited nondeterminism and the complexity of the V-C dimension. J. Comput. Syst. Sci. **53**(2) (1996) 161–170
7. Ginsburg, S.: The Mathematical Theory of Context-Free Languages. McGraw-Hill, Inc., New York, NY, USA (1966)
8. Papadimitriou, C.: Computational Complexity. Addison Wesley (1994)
9. Ladner, R.E.: On the structure of polynomial time reducibility. J. ACM **22**(1) (1975) 155–171

Formalizing Dangerous SAT Encodings

Alexander Hertel, Philipp Hertel, and Alasdair Urquhart*

Department of Computer Science,
University of Toronto, Toronto ON M5S 3G4, Canada
{ahertel,philipp,urquhart}@cs.toronto.edu

Abstract. In this paper we prove an exponential separation between two very similar and natural SAT encodings for the same problem, thereby showing that researchers must be careful when designing encodings, lest they accidentally introduce complexity into the problem being studied. This result provides a formal explanation for empirical results showing that the encoding of a problem can dramatically affect its practical solvability.

We also introduce a domain-independent framework for reasoning about the complexity added to SAT instances by their encodings. This includes the observation that while some encodings may add complexity, other encodings can actually make problems easier to solve by adding clauses which would otherwise be difficult to derive within a Resolution-based SAT-solver. Such encodings can be used as polytime preprocessing to speed up SAT algorithms.

1 Introduction

Satisfiability, or SAT, is the archetypal $\mathcal{NP}$-Complete problem. It has long been known that every problem in $\mathcal{NP}$ can be reduced to SAT using Cook's Theorem [Coo71]. Since propositional formulas are very expressive, instances of many problems in $\mathcal{NP}$ can also be encoded as SAT instances in a much more direct and intuitive way than via Cook's Theorem. This has allowed research into solving these varied problems to be concentrated on SAT-solving. In fact, many problems have numerous different SAT encodings to choose from. Building a framework for comparing the effectiveness of competing encodings is the main focus of this paper.

The strategy of translating problems from other domains to SAT has proved to be fruitful. Nevertheless, this technique is not without its dangers. Empirical evidence suggests that natural encodings which seem to conserve much of the structure of the original problem can actually convert simple instances of the original problem to very difficult SAT formulas. For example, in [KMS96] numerous approaches for translating planning problems to SAT are investigated. Some are found to result in formulas which are much harder to solve than others.

* This research supported by NSERC and the University of Toronto Department of Computer Science.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 159–172, 2007.

This suggests that a great deal of care must be taken in designing encodings since one cannot assume that they will conserve the simplicity of input instances.

This danger is so well-known to the Propositional Reasoning community that the authors of [KMS97] list understanding it as one of ten important and challenging open problems in the area. A more recent follow-up paper [KS03] reaffirms this problem's importance and notes that although some progress has been made, there is still much more work to be done.

We address this problem in two ways. Firstly, we provide a formal example of a natural encoding which translates a trivial instance of the Hamiltonian Cycle problem to an intractably difficult one for any Resolution-based SAT-solver. We also show that very minor modifications to this encoding can make it produce easy SAT instances, thereby formally proving an exponential separation between two very similar and natural encodings. Secondly, we provide a domain-independent framework for comparing the effectiveness of competing encodings, in terms of how easy it is to solve their outputs. This framework relies on the existence of a proof-system hierarchy and the close relationship between SAT-solving and propositional proof complexity.

This paper is organized as follows: In Sections 2 - 4, we provide examples of two very similar and natural SAT encodings for the $\mathcal{NP}$-Complete Hamiltonian Cycle problem, and use them to encode a family of trivially non-Hamiltonian graphs. These encodings are not pathologically designed to create problems, but rather are very intuitive and straightforward. The result of the first encoding is the family of formulas from Theorem 1, which we show has exponential lower bounds for AC^0-Frege proof systems, which immediately imply exponential lower bounds for Resolution (RES), and all RES-based SAT-solvers, including Clause Learning.

The result of the second encoding is the family of formulas from Theorem 2, which we show to have polynomial upper bounds for Tree Resolution (T-RES) / DPLL, one of the weakest Resolution-based proof systems. This shows that the results of these encodings are respectively very hard and very easy to solve in practice, even though they are very similar and came from the same graph. This result is relevant to the eighth open problem in [KMS97, KS03] because as already mentioned, the Propositional Reasoning community is aware of empirically tested instances where different reductions can have a significant impact on the complexity of a problem, but this is the first formal example.

In addition, researchers have noted empirically that adding redundant clauses to formulas can transform very difficult instances of SAT into very easy ones. We can further weaken the easy formulas from Theorem 2 to obtain the formulas of Corollary 2 which contain the clauses of the hard formula from Theorem 1 as a proper subset and still have T-RES refutations of polynomial size. This provides a formal example of hard instances which can be converted to easy instances by the addition of redundant clauses.

In Section 5, we provide a formal domain-independent framework which captures our intuitive notions of what constitute dangerous and safe reductions. We show that not only can reductions increase the complexity of problems but they

can beneficially reduce their complexity by applying polytime reasoning which is unavailable to the target proof system.

For the purposes of this paper, we assume that the reader is familiar with the basics of proof complexity and general complexity theory. We shall use [CK01] as our reference for proof complexity. There is a very close relationship between proof-complexity lower bounds and lower bounds for their corresponding algorithms. For example, DPLL is almost identical to the T-RES proof system. Since RES subsumes T-RES, and since RES has exponential lower bounds for the pigeonhole principle [Hak85], it follows that T-RES and therefore DPLL must also have exponential lower bounds for the pigeonhole formulas. In effect, no RES-based SAT-solver will ever be able to solve pigeonhole formulas in polynomial time. Since almost all SAT-solvers are based on refinements of RES, this gives us immediate lower bounds for algorithms such as DPLL and Clause Learning. We use the framework of proof-complexity to categorize encodings as being harmful, neutral, or beneficial.

2 A SAT Encoding for the Hamiltonian Cycle Problem

Consider the following Hamiltonian Cycle to SAT reduction: take a graph $G = (V, E)$ and create a formula F which enforces a mapping from V to the positions p_1, p_2, ..., $p_{|V|}$ of a Hamiltonian Cycle H. Intuitively, H can be thought of as a cyclic ordering on the $|V|$ vertices of G, where vertex i can be mapped to the j^{th} position in H if i is adjacent in G to the vertices mapped to positions $j-1$ and $j+1$. F contains variables of the form $m_{i,j}$, each of which is interpreted as meaning that element i from G (the domain) is mapped to position j in the Hamiltonian Cycle H (the range). F partially consists of clauses which enforce a bijection between V and H. A conjunction of the following groups of clauses ensure such a bijection:

Total: $\bigwedge_{i=1}^{|V|}(\bigvee_{j=1}^{|V|} m_{i,j})$ *i.e. Every vertex in V maps to at least one position in H.*

Onto: $\bigwedge_{j=1}^{|V|}(\bigvee_{i=1}^{|V|} m_{i,j})$ *i.e. Every position in H has at least one vertex mapped to it.*

1-1: $\bigwedge_{j=1}^{|V|}\bigwedge_{i_1=1}^{|V|}\bigwedge_{\substack{i_2=1\\ i_1\neq i_2}}^{|V|}(\neg m_{i_1,j} \vee \neg m_{i_2,j})$ *i.e. At most one vertex maps to each position.*

Fn.: $\bigwedge_{i=1}^{|V|}\bigwedge_{j_1=1}^{|V|}\bigwedge_{\substack{j_2=1\\ j_1\neq j_2}}^{|V|}(\neg m_{i,j_1} \vee \neg m_{i,j_2})$ *i.e. Every vertex maps to at most one position.*

To ensure that F is satisfiable if and only if G is Hamiltonian, we need only add the following clauses which place constraints on the bijection corresponding to the structure of G:

$$\text{Edge: } \bigwedge_{j=1}^{|V|} \bigwedge_{i=1}^{|V|} \bigwedge_{\substack{k:(i,k)\notin E \\ i\neq k}} (\neg m_{i,j} \vee \neg m_{k,(j+1) \bmod |V|})$$

Informally, the edge constraint clauses are ensuring that for every non-edge (i, k), if vertex i has been mapped to the j^{th} position in the cycle H, then vertex k cannot be mapped to position $j + 1 \pmod{|V|}$. It is not hard to see that the reduction is correct: if G is Hamiltonian, then these edge constraints will not cause a contradiction with the clauses enforcing the bijection, so F will be satisfiable. Likewise, if F is satisfiable, then it means that there is a bijection from V to H which respects the constraints enforced by the edge clauses, so G must be Hamiltonian.

Of course, the total, onto, 1-1, and function clauses are more than enough to ensure a bijection. In fact, the total and 1-1 clauses by themselves are sufficient, as are the onto and function clauses by themselves. This leads us to define some notation. Let $H(G)$ be the formula resulting from the above reduction. To this we add a subscript showing which clause groups were used in its construction. We abbreviate total as T, onto as O, 1-1 as 1, and function as F. For example, if we used clauses from the total and 1-1 groups, then the formula is labeled as $H(G)_{T,1}$. There is no need to specify that edge clauses were used, because all of our encodings require them.

An Interesting Family of Graphs

Consider the complete graph on n vertices, K_n. Let K_n^* be K_n with the addition of a single degree-0 vertex. We shall apply the reductions from the previous section to graphs from this family. Since each K_n^* is disconnected, it is trivially non-Hamiltonian, which in turn means that every formula $H(K_n^*)$ is unsatisfiable.

We will show that K_n^* is interesting because proofs of $H(K_n^*)$ either have polynomial upper bounds for T-RES (and therefore all stronger systems), or exponential lower bounds for AC^0-Frege (and therefore all weaker systems), depending on which clauses are used in its construction. This provides us with a formal example of two very similar and natural encodings whose outputs have drastically different complexities.

3 Exponential Lower Bounds for $H(K_n^*)_{T,1,F}$

In this section we show that the version of the encoding which uses Total, 1-1, and Function clauses, when applied to K_n^* graphs, results in a formula which no Resolution-based SAT-solver can efficiently solve, even though the K_n^* graphs are trivially non-Hamiltonian. In other words, even though the encoding is very natural, it injects an exponential amount of unwanted complexity into our original problem instance.

Theorem 1. *Lengths of* AC^0-Frege *proofs for the unsatisfiability of* $H(K_n^*)_{T,1,F}$ *formulas have* $\Omega(2^{\sqrt[5^d]{n}})$ *lower bounds, where* d *is the depth of the Frege proof, and if there exist size-*N AC^0-Frege *proofs restricted by* $m_{x,n} = 1$ *of* $H(K_n^*)_{T,1,F}$*, then there exist size-*$N + O(n^3)$ *proofs of* $fPHP_{n-2}^n$.

Proof: The high-level overview of this proof is as follows: Assume that we have a size-N AC^0-Frege proof of $H(K_n^*)$. We show that this proof can be restricted with a specially-chosen truth assignment α to get a new, smaller proof of $H(K_n^*) \restriction_\alpha$. After unit propagation, this formula becomes $fPHP_{n-2}^n$ which is already known to have exponential AC^0-Frege lower bounds. For those unfamiliar with AC^0-Frege, it suffices to think of this proof in terms of RES, ie. any size-N RES proof of $H(K_n^*)$ can also be restricted to yield $fPHP_{n-2}^n$. Since the lower bound is proved for AC^0-Frege proof systems, we show how to model the unit propagations using $O(n^3)$ steps of AC^0-Frege reasoning. Therefore, if there exists a sub-exponential AC^0-Frege (resp. RES) proof of $H(K_n^*) \restriction_\alpha$, then there exists a sub-exponential AC^0-Frege (resp. RES) proof of $fPHP_{n-2}^n$, which is a contradiction, since $fPHP_{n-2}^n$ has exponential lower bounds [BT88].

The details of the proof are as follows: The restriction that we apply to $H(K_n^*)_{T,1,F}$ is $m_{x,n} = 1$. Intuitively, this will guarantee via the edge clauses that we cannot map any vertex to positions $n-1$ or $n+1$ because x has no edges incident on it. If we interpret the variables as mappings from pigeons to holes, we now have two more pigeons than holes. The restriction $m_{x,n} = 1$ propagates as follows:

- For every function clause of the form $(\neg m_{x,n} \vee \neg m_{x,j})$, since we have set $m_{x,n}$ to 1, we must set all of $m_{x,1}, m_{x,2}, ..., m_{x,n-1}$ as well as $m_{x,n+1}$ to 0.
- For every 1-1 clause of the form $(\neg m_{x,n} \vee \neg m_{i,n})$, propagating $m_{x,n} = 1$ causes us to set all of $m_{1,n}, m_{2,n}, ..., m_{n,n}$ to 0.
- Finally, for every edge clause of the form $(\neg m_{x,n} \vee \neg m_{k,n+1})$ where (x,k) is a non-edge in G, propagating $m_{x,n} = 1$ causes us to set all of $m_{1,n+1}, m_{2,n+1}$, $..., m_{n,n+1}$ to 0. Similarly, for each edge clause of the form $(\neg m_{i,n-1} \vee \neg m_{x,n})$, propagating causes us to set all of $m_{1,n-1}, m_{2,n-1}, ..., m_{n,n-1}$ to 0.

The effect of these propagations on the various groups is as follows:

- Total Clauses: The restriction $m_{x,n} = 1$ satisfies the clause $(m_{x,1} \vee m_{x,2} \vee ... \vee m_{x,n} \vee m_{x,n+1})$. Combined with this, the propagations $m_{i,n-1} = 0$, $m_{i,n} = 0$, and $m_{i,n+1} = 0$ for all i causes the total clauses to become:
  $\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{n-2} m_{i,j})$
- 1-1 Clauses: For each $1 \leq i \leq n+1, i \neq x$, there is a clause $(\neg m_{x,n} \vee \neg m_{i,n})$. Since every $m_{i,n}$ was set to 0, every 1-1 clause involving any $m_{i,n}$ will be satisfied and eliminated. Due to the edge clause propagations, for every $i \neq x$, every clause involving $m_{i,n-1}$ or $m_{i,n+1}$ will also be eliminated. The 1-1 clauses therefore become:
  $\bigwedge_{j=1}^{n-2} \bigwedge_{i_1=1}^{n} \bigwedge_{\substack{i_2=1 \\ i_2 \neq i_1}}^{n} (\neg m_{i_1,j} \vee \neg m_{i_2,j})$

- Function Clauses: For each $1 \leq j \leq n+1, j \neq n$ there is a clause $(\neg m_{x,n} \vee \neg m_{x,j})$. Since every $m_{x,j}$ was set to 0, every function clause involving any $m_{x,j}$ will be satisfied and eliminated. Due to edge clause propagations, for every $i \neq x$, every clause involving $m_{i,n-1}$ or $m_{i,n+1}$ will also be eliminated. Due to the 1-1 clause propagations, for every $i \neq x$, every clause/ involving $m_{i,n}$ will also be eliminated. The function clauses therefore become:
$\bigwedge_{i=1}^{n} \bigwedge_{j_1=1}^{n-2} \bigwedge_{\substack{j_2=1 \\ j_2 \neq j_1}}^{n-2} (\neg m_{i,j_1} \vee \neg m_{i,j_2})$
- Edge Clauses: There are two types of edge clauses: those which contain the literal $\neg m_{x,n}$, and those which contain the literal $\neg m_{x,j}, j \neq n$. Note that this covers all edge clauses because vertex x is involved in every non-edge of K_n^*. Clauses of the first type are satisfied by unit propagation which forces the other literal in each such clause to be set to 1. Those of the second type are satisfied by the $m_{x,j}$ propagations from the function clauses. All edge clauses are therefore eliminated.

These remaining clause groups when simplified by unit propagation are exactly the clauses from $fPHP_{n-2}^n$. In effect, the restricted proof of $H(K_n^*)_{T,1,F}$ has been turned into a proof of $fPHP_{n-2}^n$. It is not hard to show that $\mathsf{AC^0}$-Frege can perform unit propagations in polynomial size for some polynomial $p(n)$. This turns our size-N proof of $H(K_n^*)_{T,1,F}$ to a size $N + p(n)$ proof of $fPHP_{n-2}^n$.

Let d be the depth bound imposed on a Frege system. Since $\mathsf{AC^0}$-Frege proof systems are closed under restriction (ie. restricting a proof yields a smaller proof), and since they have $\Omega(2^{\sqrt[5d]{n}})$ size lower bounds for $fPHP_{n-2}^n$ formulas [UF96], we may conclude that the $H(K_n^*)_{T,1,F}$ formulas also require proofs of size at least $\Omega(2^{\sqrt[5d]{n}})$. Specifically, if there exist size-N $\mathsf{AC^0}$-Frege proofs restricted by $m_{x,n} = 1$ of $H(K_n^*)_{T,1,F}$, then there exist size-$N + p(n)$ proofs of $fPHP_{n-2}^n$. □

Clearly, this result holds for all formulas such as $H(K_n^*)_{T,1}$ which are composed of proper subsets of the clauses from $H(K_n^*)_{T,1,F}$.

Corollary 1. *No SAT algorithm based on* $\mathsf{AC^0}$-Frege *nor any weaker proof system can efficiently solve* $H(K_n^*)_{T,1,F}$ *formulas. This includes* DPLL *as well as Clause-Learning based SAT-solver algorithms.*

Therefore we have shown that the $H(G)_{T,1,F}$ encoding can convert trivial instances of the Hamiltonian Cycle problem to intractable SAT instances.

4 Polynomial Upper Bounds for $H(K_n^*)_{T,O,F}$

In this section we show that the version of the encoding which uses Total, Onto, and Function clauses, when applied to K_n^* graphs, results in a formula which has short DPLL proofs. This is particularly interesting because both $H(K_n^*)_{T,O,F}$ and $H(K_n^*)_{T,1,F}$ are natural encodings of the Hamiltonian Cycle problem, and neither is a subset of the clauses of the other, but $H(K_n^*)_{T,O,F}$ is easy to solve, while $H(K_n^*)_{T,1,F}$ is intractably difficult.

Theorem 2. T-RES *proofs for the unsatisfiability of* $H(K_n^*)_{T,O,F}$ *formulas have* $O(n^2)$ *size upper bounds, where* n *is the number of distinct variables contained in the formulas.*

Proof: For the following argument, please refer to the T-RES proof template shown in Figure 1. Note that some of the leaves are labelled with the clauses which are falsified at that position. In addition, the remaining leaves are labelled with the groups containing the clauses which are falsified.

We initially branch on $m_{x,1}$. Since x is an isolated vertex in K_n^*, setting $m_{x,1} = 1$ ensures that assigning any future vertex to position 2 (ie. setting $m_{i,2} = 1$ for any $i \neq x$) will falsify an edge clause, and therefore falsify the formula. Setting $m_{x,2} = 1$ also falsifies $H(K_n^*)_{T,O,F}$ by falsifying a function clause, because no vertex may be assigned to more than one position. Finally, setting $m_{i,2} = 0$ for $i = 1, 2, ..., n, x$ will falsify the onto clause requiring that some vertex be mapped to position 2. This subtree requires $2n + 3$ nodes.

When we set $m_{x,1} = 0$, we next branch on $m_{x,2}$. Clearly, the formula rooted by setting $m_{x,2} = 1$, can be shown to be unsatisfiable with a tree of size $2n + 3$ for the same reasons as above.

For each i, after setting $m_{x,i-1} = 0$, we branch on $m_{x,i}$. Each positive branching will result in a subtree of size $2n + 3$ as described above.

The all-negative assignment to every $m_{x,i}$ falsifies the total clause ensuring that vertex x is mapped to some position, after which all branching is complete. This T-RES proof therefore has size $(n + 1)(2n + 3) + 1$, which is $O(n^2)$, as required. □

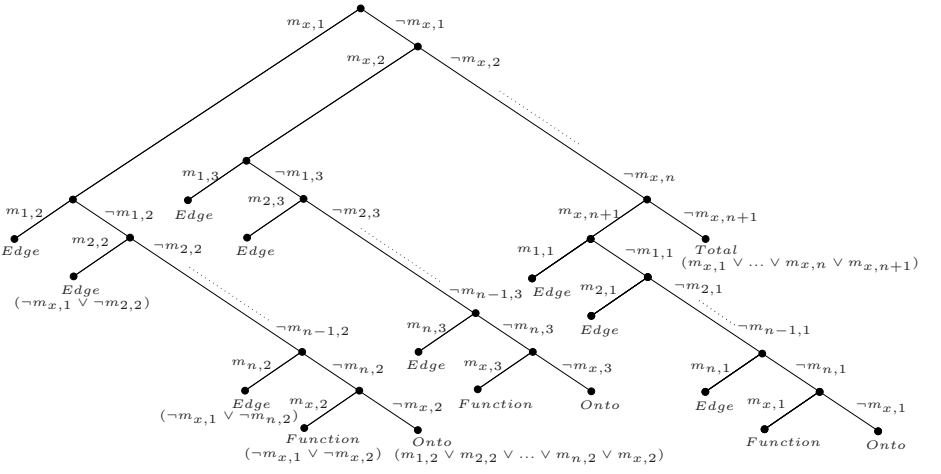


Fig. 1. A Template for Polynomially-Sized T-RES Proofs for the Unsatisfiability of $H(K_n^*)_{T,O,F}$ Formulas

Corollary 2. *The size of* T-RES *proofs for the unsatisfiability of* $H(K_n^*)_{T,O,1,F}$ *formulas have polynomial upper bounds.*

Proof: The same T-RES proof which proves the unsatisfiability of $H(K_n^*)_{T,O,F}$ proves the unsatisfiability of $H(K_n^*)_{T,O,1,F}$. □

Corollary 3. *For any* $H(K_n^*)_{T,O,1,F}$ *formula, there is a polynomially-bounded* DPLL *computation which solves it.*

5 Domain Independent Framework for Comparing Encodings

Currently, no system exists to classify encodings according to whether they make problem instances harder or easier. Such a classification system might prove to be very beneficial for researchers who are actively using SAT-solvers to tackle $\mathcal{NP}$-Complete problems. It might also prove to be beneficial for researchers who are interested in studying the phenomenon of dangerous encodings more abstractly with an eye to finding general principles for predicting which encodings will lead to complexity blow-ups on certain families of formulas. In this section, we provide a framework for such a system.

Although no system for classifying encodings has yet been devised, a lot of work has gone into classifying the power of proof systems. These proof systems have been organized into a hierarchy based on polynomial simulations and exponential separations. Briefly, a proof system α is said to p-simulate another proof system β if for every unsatisfiable CNF formula f, there exists an α refutation of f which is at most a polynomial factor larger than f's smallest β refutation. A proof system α is said to be exponentially separated from another proof system β if there exists some class of formulas F such that for all $f \in F$ there exists an α refutation of f with polynomial size, but the smallest β refutation of f has exponential size. Such a separation clearly implies that β cannot p-simulate α. If α p-simulates β and is also exponentially separated from β, then we say that α is strictly stronger than β, and there is always a (nondeterministically chosen) computation of a SAT-solving algorithm based on the principles of α which will finish within a polynomial factor of the time it would take any SAT-solver based on the principles of β to finish.

Much work has gone into establishing a proof system hierarchy especially for systems based on the Resolution rule. Please refer to Figure 2 below for the portion of the hierarchy which is particularly relevant to propositional reasoning and SAT solving.

Each node in the diagram represents all of the families of formulas which have polynomial size refutations in the system labeling the node. Arrows represent p-simulation relationships between systems. An arrow from system α to system β means that α p-simulates β. A slash through an arrow from α to β represents an exponential separation between β and α. An arrow labeled with a question mark denotes an unknown relationship. Systems to the left in the diagram are generally stronger than systems to the right. Though short refutations exist for

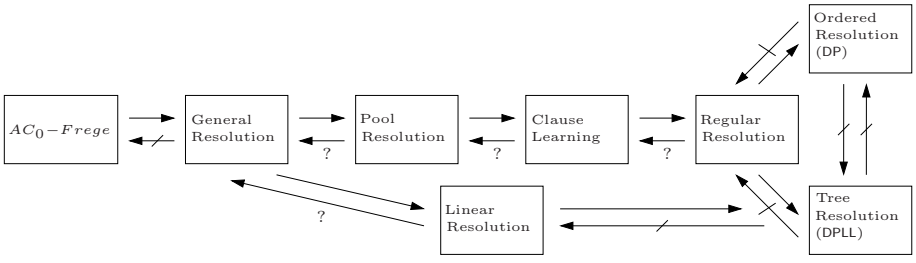


Fig. 2. Part of the Proof Complexity Hierarchy

larger classes of formulas in stronger systems, finding them is generally more difficult than finding refutations in weaker systems. Hence SAT-solvers based on the Resolution rule generally do not have the full power of RES, but instead implement some form of DPLL which is equivalent to T-RES and is very low in the hierarchy. It is therefore desirable for instances which we want to solve to exist in nodes that are low down in the hierarchy. This gives us a better chance of deterministically finding a refutation in a short amount of time.

We can use this hierarchy to judge the quality of SAT encodings. If some input to an encoding is at one level of the hierarchy and its corresponding output only exists in higher levels, then the encoding is dangerous with respect to that input since its result requires more power to solve. If the input and output of an encoding exist in all of the same levels of the hierarchy, then the encoding is neutral with respect to that input. If some input to an encoding exists nowhere below a certain level in the hierarchy, but its output does, then that encoding actually makes a potentially exponential contribution towards solving the instance. Since every encoding only takes polynomial time to compute, such beneficial encodings can be used as efficient preprocessing steps and identifying them is of great practical interest. We coin the terms *explosive*, *stable*, and *implosive* to refer to encodings which are harmful, neutral, and beneficial with respect to certain families of formulas and certain proof systems. These are defined formally below and examples of each are given.

5.1 Explosivity

Definition 1. *Let α be a proof system for a language L_1, let β be a proof system for a language L_2, and let $R : L_1 \to L_2$ be a reduction from L_1 to L_2. If there exists some family of strings $X = \{x_1, x_2, ...\}$, $X \subseteq L_1$ such that for all k and for all $x_i \in X$ there exists an α-proof P_1 of x_i, but there exists no β-proof P_2 of $R(x_i)$ such that $|P_2| \leq |P_1|^k$, then we say that the reduction R is (α, β)-Explosive on the set X.*

This definition corresponds to our intuitive notion of what constitutes a dangerous reduction, and we can immediately apply it to our main result:

Corollary 4. *The Hamiltonian Cycle to SAT reduction above which uses the T, 1, and F clauses is (α,$\mathsf{AC^0}$-Frege)-Explosive on the set containing the K_n^* graphs*

for any non-Hamiltonicity proof system α which has polynomially-bounded proofs of the K_n^* graphs.

An example of such a non-Hamiltonicity proof system is NHPS, given in [Her06]. This example is interesting, since tree-like NHPS seems to be weaker than RES, let alone any AC^0-Frege system. We therefore have an example of a reduction that injects enough complexity to send its outputs' difficulty several levels up the proof complexity hierarchy.

Another formal example of Explosivity comes from a corollary of the main result of [HU06a] which proves that the reduction from QBF to Intuitionistic Propositional Logic (IPL) given by Statman in [Sta79] is probably Explosive:

Corollary 5. *Unless $\mathcal{NP} = co\mathcal{NP}$, Statman's reduction is (α,LJ[$\boldsymbol{E_S}$])-Explosive for any QBF proof system α which has polynomially-bounded proofs for any prenex instance of the law of excluded middle (i.e. formulas of the form $p \vee \neg p$).*

Explosivity is caused when an encoding increases the proof complexity of the input instance. In the case of RES, if a reduction fails to introduce clauses which are needed in order to provide a short RES proof, then the reduction is Explosive, and there is no hope of solving the translation. The 'onto' clauses discussed in Section 2 are an example of such clauses which have no short RES derivations themselves and can make an exponential difference to the proof complexity of the reduction's output.

Those interested in proof complexity will note that Explosivity is trivially associated with exponential separations between proof systems. Every example of p-simulation between two proof systems on the same language for which there is a superpolynomial separation implicitly gives an example of (α, β)-Explosivity. If proof system β p-simulates proof system α, but α can not p-simulate β, then the trivial reduction of doing nothing is (α, β)-Explosive on the set of formulas which provides the separation. For this reason, (α, β)-Explosive reductions where α is a strictly stronger proof system than β (for example, (AC^0-Frege, T-RES)-Explosive reductions) are not nearly as interesting as (α, β)-Explosive reductions in which α is a strictly weaker proof system than β.

5.2 Stability

Definition 2. *Let α, β, L_1, L_2, and R be as in Definition 1. If there exist constants k_1 and k_2 and a family of strings $X = \{x_1, x_2, ...\}$, $X \subseteq L_1$ such that for any α-proof P_1 of x_i there exists a β-proof P_2 of $R(x_i)$ where $|P_2| \leq |P_1|^{k_1}$ and $|P_1| \leq |P_2|^{k_2}$ then we say that the reduction R is (α, β)-Stable on the set X.*

From a proof-complexity point of view, every example of p-equivalence implicitly gives an example of (α, β)-Stability. If α and β are two p-equivalent proof systems for the same language L, then the trivial reduction of doing nothing is both (α, β)-Stable and (β, α)-Stable for the entire language L. For this reason, (α, β)-Stable reductions for p-equivalent proof systems are not nearly as interesting as ones for proof systems for which there is a superpolynomial separation.

A more interesting example of stability not associated with p-equivalence is the relationship between RES and Linear Resolution (L-RES) given in [BOP03]. More specifically, the authors provide a very simple reduction R which consists of adding trivial clauses of the form $(p \vee \neg p)$ for each variable p of the original formula, and show that for every RES proof, there exists an L-RES proof of $R(w)$ which is only polynomially larger. In other words, R is (RES,L-RES)-Stable on the entire SAT language.

Another example of stability comes from Theorem 2 above:

Corollary 6. *The Hamiltonian Cycle to SAT reduction above which uses the T, O, 1, and F clauses is (α,T-RES)-Stable on the set of K_n^* graphs for any non-Hamiltonicity proof system α which has polynomially-bounded proofs for the K_n^* graphs.*

As already mentioned, NHPS from [Her06] is such an α.

5.3 Implosivity

In practical terms, an even more helpful characteristic for encodings is that of Implosivity. Intuitively, an encoding which takes hard formulas for one proof system and converts them into easy ones for another is Implosive. In other words, Implosive reductions can make otherwise hard instances more accessible to SAT-solvers. Examples of such beneficial reductions are already known to the Propositional Reasoning community; it has been shown that SAT encodings of Constraint Satisfiability Problem (CSP) instances can be optimized with respect to local consistency checking and unit propagation [KS03]. In this case the reduction from CSP to SAT actually has beneficial properties, namely that it reduces the proof complexity of its inputs with respect to the consistency conditions. Another good example of this phenomenon is shown in [BB03], where an encoding is provided that transforms the parity problem, which for many years was considered to be a hard DIMACS instance, into formulas that are easy for DPLL-based solvers.

More formally, the beneficial property of Implosivity is defined as follows:

Definition 3. *Let α, β, L_1, L_2, and R be as in Definition 1. If there exists some family of strings $X = \{x_1, x_2, ...\}$, $X \subseteq L_1$ such that for all k and for all $x_i \in X$ there exists a β-proof P_2 of $R(x_i)$ but there exists no α-proof P_1 of x_i such that $|P_1| \leq |P_2|^k$, then we say that the reduction R is (α, β)-Implosive on the set X.*

As with Explosivity, Implosivity is trivially associated with p-simulation. Every example of p-simulation between two proof systems on the same language for which there is a superpolynomial separation implicitly gives an example of (α, β)-Implosivity. If proof system β p-simulates proof system α, but α can not p-simulate β, then the trivial reduction of doing nothing is (α, β)-Implosive on the set of formulas which gives the separation. For this reason, (α, β)-Implosive reductions where α is a strictly weaker proof system than β

(for example, (T-RES, AC^0-Frege)-Implosive reductions) are not nearly as interesting as (α, β)-Implosive reductions in which α is strictly stronger than β.

Again, a non-trivial example of Implosivity comes from the NHPS proof system. Let $G_{\frac{n}{2},\frac{n}{2}}$ be the graph consisting of two disjoint cliques of size $\frac{n}{2}$. These graphs have exponential NHPS lower bounds [Her06]. However, the formulas resulting from applying the reduction from Section 2 which uses the T, O, 1, and F clauses have polynomial T-RES upper bounds [HU06b]. In other words, this reduction is (NHPS, T-RES)-Implosive on the $G_{\frac{n}{2},\frac{n}{2}}$ graphs, which is interesting because the T, 1, F version of the reduction from Corollary 4 is (NHPS,AC^0-Frege)-Explosive on the K_n^* graphs. This gives a clear example of how different inputs to the same system can be simplified or complicated depending on encoding.

An interesting potential example of Implosivity is the L-RES reduction R from [BOP03] mentioned above. As already stated, R is (RES,L-RES)-Stable on the entire SAT language. However, it is unknown whether there is an exponential separation between L-RES and RES. If so, then R is (L-RES,L-RES)-Implosive on the inputs which give the separation. Such examples of reflexive implosivity are good candidates for beneficial preprocessing.

Generally speaking, non-trivial Implosivity arises when polytime reductions make use of reasoning which is not available to their target proof system. A polytime reduction in RES might add clauses to the instance which could not otherwise be derived concisely in RES. Given these clauses, Resolution-based solvers can easily solve the problem, but without them, they require exponential time. In effect, such reductions allow solvers to 'cheat' and do work that cannot be done by their underlying proof systems.

5.4 Alternate Hierarchies

Though this the proof system hierarchy may prove to be useful for classifying encodings, we could also produce alternative hierarchies for which the notions of explosivity, stability, and implosivity could be used to classify encodings. In order to use the proof system hierarchy for this task we need to perform a fairly robust analysis of the family of problem instances being studied, as we did in Sections 3 & 4. A more empirical hierarchy based on the real world performance of specific implementations on families of inputs may be preferred.

6 Implications for Proof Complexity

Whenever the relationship between two proof systems on different languages is studied, there must necessarily be a reduction involved. Weak proof systems such as RES and its refinements are not powerful enough to perform polytime reductions. This necessitates the use of a separate polytime algorithm to perform the reduction. Since the details of the reduction can affect the proof complexity of its output, it does not make sense to talk about p-simulation or exponential separation between two weak proof systems over different languages. Rather, one

must talk about p-simulation or exponential separation *with respect to a specific reduction*. If a reduction exists which allows one proof system to p-simulate another, we say that the first proof system effectively p-simulates the second. We formally define this notion as follows.

Definition 4. *Let $f_1 : S_1^* \to L$ and $f_2 : S_2^* \to L$ be proof systems. If there exists a k and a polytime reduction $r : L_1 \to L_2$ such that $y \in L_1$ if and only if $r(y) \in L_2$ and for all $x_1 \in S_1^*$ there exists an $x_2 \in S_2^*$ such that $r(f_1(x_1)) = f_2(x_2)$ and $|x_2| \leq |x_1|^k$, then we say that f_2 effectively polynomially-simulates f_1.*

If there also exists a polytime computable function $t : S_1^* \to S_2^*$ such that for all $x \in S_1^* r(f_1(x)) = f_2(t(x))$, then f_2 effectively p-simulates f_1.

We can just as easily consider this definition applied to two proof systems over the same language. This yields a generalization of the normal notion of p-simulation. For example, though L-RES is not known to p-simulates RES, it does effectively p-simulate RES since the polytime reduction in [BOP03] is (RES,L-RES)-Stable on the entire SAT language.

7 Concluding Remarks

The idea that encodings can inject complexity into a problem is disconcerting. It is worrisome to think that a reduction from one problem to another can negatively affect the proof complexity of the result and potentially make the instance difficult for proof systems which are located several levels higher in the proof complexity hierarchy than the intended system. Furthermore, as we have shown in this paper, this phenomenon can happen with very natural and even obvious encodings. Even more worrisome is that it does not seem to be at all obvious which types of reductions have this property. With our example, we were lucky enough to see that the input graph was translated to a formula which is very similar to the pigeonhole formulas, but in general we cannot expect to be so lucky. There are probably infinitely many families of formulas which have no short RES proofs and it would not be easy to identify them lurking within the output of an encoding. Random formulas, which are very hard to categorize, as well as other combinatorial problems which have never even been investigated could act very much like the pigeonhole formulas do in our example. If we do not even know what these formulas look like, then it is probably very difficult to predict and avoid reductions which might produce them or something similar to them. Further research is needed in order to characterize which types of reductions have this property.

As a first step towards a characterization, we have outlined a framework for comparing encodings based on the proof complexity hierarchy. The key idea behind the framework is that encodings can affect the proof complexity of the result either beneficially or adversely by overcoming the superpolynomial separation between two proof systems through the use of reasoning that is unavailable to the proof system or by requiring the proof system to derive clauses which cannot be derived concisely.

Acknowledgements

We would like to thank Fahiem Bacchus and Toni Pitassi for their many valuable comments and suggestions regarding this paper.

References

[BB03] O. Bailleux and Y. Boufkhad. Efficient CNF Encodings of Boolean Cardinality Constraints. *International Conference on the Principles and Practice of Constraint Programming*, pages 102 – 122, 2003.

[BOP03] J. Buresh-Oppenheim and T. Pitassi. The Complexity of Resolution Refinements. *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, 2003.

[BT88] S. Buss and G. Turán. Resolution Proofs of Generalized Pigeonhole Principles. *Theoretical Computer Science*, 62:311 – 317, 1988.

[CK01] P. Clote and E. Kranakis. *Boolean Functions and Computation Models*. Springer-Verlag, Berlin, 2001.

[Coo71] S.A. Cook. The Complexity of Theorem-Proving Procedures. *Proceedings of the Third Annual ACM Symposium on the Theory of Computation*, pages 151 – 158, 1971.

[Hak85] A. Haken. The Intractability of Resolution. *Theoretical Computer Science*, 39:297 – 308, 1985.

[Her06] A. Hertel. A Non-Hamiltonicity Proof System. Unpublished Manuscript, 2006.

[HU06a] A. Hertel and A. Urquhart. Proof Complexity of Intuitionistic Propositional Logic. Unpublished Manuscript, 2006.

[HU06b] A. Hertel and A. Urquhart. Prover / Delayer Game Upper Bounds For Tree Resolution. Unpublished Manuscript, 2006.

[KMS96] H. Kautz, D. McAllester, and B. Selman. Encoding Plans in Propositional Logic. *Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning*, 1996.

[KMS97] H. Kautz, D. McAllester, and B. Selman. Ten Challenges in Propositional Reasoning and Search. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[KS03] H. Kautz and B. Selman. Ten Challenges Redux: Recent Progress in Propositional Reasoning and Search. *Ninth International Conference on Principles and Practice of Constraint Programming*, 2003.

[Sta79] R. Statman. Intuitionistic Propositional Logic is Polynomial-Space Complete. *Theoretical Computer Science*, 9:67 – 72, 1979.

[UF96] A. Urquhart and X. Fu. Simplified Lower Bounds for Propositional Proofs. *Notre Dame Journal of Formal Logic*, 37:523 – 545, 1996.

Algorithms for Variable-Weighted 2-SAT and Dual Problems

Stefan Porschen and Ewald Speckenmeyer

Institut für Informatik, Universität zu Köln,
Pohligstr. 1, D-50969 Köln, Germany
{porschen,esp}@informatik.uni-koeln.de

Abstract. In this paper we study NP-hard variable-weighted satisfiability optimization problems for the class 2-CNF providing worst-case upper time bounds holding for arbitrary real-valued weights. Moreover, we consider the monotone dual class consisting of clause sets where all variables occur at most twice. We show that weighted SAT, XSAT and NAESAT optimization problems for this class are polynomial time solvable using appropriate reductions to specific polynomial time solvable graph problems.

Keywords: weighted satisfiability, optimization problem, NP-hardness, edge cover, graph factor, perfect matching.

1 Introduction

Weighted variants of search or decision problems are of certain importance for computational complexity theory as they can provide a gap from easy to hard. Consider, e.g., the satisfiability problem for propositional 2-CNF formulas (2-SAT). As is well known 2-SAT can be decided in linear time and in positive case even a model for an input formula C can be found in linear time [2]. But asking for a minimum cardinality model of C, i.e., a model of least number of variables assigned to true is an NP-hard optimization problem: 2-SAT can be regarded as a generalization of the minimum vertex cover problem in undirected graphs (see below). An immediate generalization of minimum cardinality 2-SAT is minimum weight 2-SAT, where the variables of the input formula are equipped with real-valued weights. Clearly, if each variable has weight 1, we obtain the minimum cardinality problem. As shall be seen below, also the problem of maximum weight 2-SAT is NP-hard.

Weighted versions of satisfiability problems have applications e.g. in code generation where certain problems can be encoded in weighted satisfiability [1].

In this paper we address the NP-hard optimization problems minimum and maximum weight SAT for arbitrarily variable-weighted 2-CNF formulas. We provide worst-case upper bounds of $O(2^{0.5284n})$ for these problems extending results presented in [18,20] only holding for minimum weight 2-SAT, and for *non-negative* variable weights only, respectively, for maximum weight 2-SAT, and for *non-positive* variable weights only. The latter results are based on techniques

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 173–186, 2007.

provided for *mixed Horn formulas* introduced and studied in [19]. The main variants of satisfiability, namely exact satisfiability (XSAT) and not-all-equal satisfiability (NAESAT) for weighted 2-CNF have been shown to be linear time solvable in [13]. Moreover, the optimization cases of XSAT for variable weighted formulas have been shown to be solvable in time $O(2^{0.2441n})$, for arbitrary CNF [14,16], respectively, in time $O(2^{0.16254n})$ restricted to 3-CNF [11,12]. Exactly solving NAESAT for arbitrary CNF formulas (or restricted to 3-CNF) exactly in less than the trivial 2^n steps remains an open problem for the decision problem as well as for its NP-hard variable-weighted optimization variants.

Recently also counting versions of weighted SAT have been considered: #XSAT for variable weighted CNF formulas can be solved in time $O(n^2 \cdot \|C\| + 2^{0.40567 \cdot n})$ as shown in [15,16]. Fürer et al. [5] provided an algorithm for counting all maximum weight solutions of SAT for variable weighted 2-CNF formulas. Clearly, only counting models cannot provide a solution of the underlying optimization problem, as no solutions are generated explicitly.

We also consider the monotone class $\mathrm{CNF}_+(\leq 2)$, containing clauses of arbitrary length, but each variable occurs in at most two distinct clauses. We show that the variable-weighted optimization versions of SAT, XSAT, and NAESAT restricted to $\mathrm{CNF}_+(\leq 2)$ all are solvable in polynomial time via reductions to specific graph problems. Only for the XSAT cases we even obtain the same results also for the *whole* class $\mathrm{CNF}(\leq 2)$ as they appear to be special cases of results already existing.

Organisation of the paper: Section 2 describes basic definitions and terminology used throughout, followed by explaining useful monotonization tools in Section 3. Section 4 discusses optimum weight 2-Satisfiability. In Section 5 we provide polynomial time algorithms for the weighted optimizations of satisfiability and its variants for the dual class of 2-CNF. In Section 6, we finish with some open problems and concluding remarks.

2 Preliminaries

To fix notation, a *literal* is a propositional variable $x \in \{0,1\}$ or its negation $\overline{x} := \neg x$ (negated variable). The *complement* of a literal l is $\overline{l}$. A *clause* c is the disjunction of different literals and is represented as a literal set. A CNF formula C is a conjunction of different clauses and is represented as a clause set. Throughout we use the term *formula* meaning a clause set as defined. For a given formula C, clause c, by $V(C), V(c)$ we denote the set of variables contained in C, c, respectively. Similarly, given a literal l, $V(l)$ denotes the underlying variable. $V_+(C)$ (resp. $V_-(C)$) denotes the set of all variables occuring unnegated (resp. negated) in C. We distinguish between the length $\|C\|$ of a formula C and the number $|C|$ of its clauses. Let CNF denote the set of all formulas and let CNF_+ denote the set of *positive monotone* formulas, i.e., each clause contains only variables, no negated variables. Recall that the *intersection graph* of a monotone formula C has a vertex for each clause and an edge for each two intersecting clauses. For each $x \in V(C)$, let $C(x) := \{c \in C : x \in V(c)\}$. Let $\mathrm{CNF}(\leq k)$ be

the set of formulas C such that each $x \in V(C)$ occurs in at most k clauses of C regardless whether negated or unnegated. Let k-CNF denote the set of formulas C such that none of its clauses contains more than k literals. A *Horn* formula is a member of CNF such that each clause has at most one positive literal.

For $X \subseteq V(C)$, we denote by C^X the formula obtained from C by complementing exactly those literals l in C with $V(l) \in X$, and we write $C^x := C^{\{x\}}$, for simplicity. Similarly, for a truth assignment t of C let t^X be obtained from t by complementing exactly the values $t(x)$ for all $x \in X$. Again we write $t^x := t^{\{x\}}$. For $C \in$ 2-CNF, we denote by $P(C)$ its positive monotone part, i.e., the collection of exactly all positive monotone clauses in C.

The satisfiability problem (SAT) asks in its *decision* version, whether there is a truth assignment $t : V(C) \to \{0, 1\}$ assigning one literal in each clause of C to 1; such a truth assignment is called a *model of* C. SAT is known to be NP-complete [3]. In the *search* version one has to decide whether $C \in$ SAT and in the positive case one has to find a model t of C.

*Exact satisfiability (*XSAT*)* means to find a truth assignment that assigns exactly one literal in each clause of a formula to 1, called *x-model* or XSAT-*model*. *Not-all-equal satisfiability* (NAESAT) searches for a truth assignment assigning at least one literal in each clause of C to 1 and at least one literal to 0, called *nae-model* or NAESAT-*model*. The decision versions of XSAT and NAESAT are defined analogously, and are known to be NP-complete [21].

An optimization variant of SAT is obtained when weights are assigned to the variables: Given $C \in$ CNF and $w : V(C) \to \mathbb{R}$, MINW-SAT asks whether $C \in$ SAT and in the positive case one has to find a *minimum model* of C, i.e., a model t of the least weight among all models of C. The weight of a model t is defined by $w(t) = \sum_{x \in t^{-1}(1)} w(x) = \sum_{x \in V(C)} w(x)t(x)$. Analogously, the optimization problems MINW-XSAT and MINW-NAESAT are defined, which all are NP-hard for the class CNF. Similarly we obtain the maximization versions MAXW-Π, when searching for a maximum Π-model, for $\Pi \in \{\text{SAT}, \text{XSAT}, \text{NAESAT}\}$, correspondingly.

Given $M \subseteq \mathbb{R}$, let an *M-weighted formula* be a pair (C, w) where $C \in$ CNF and $w : V(C) \to M$, and let $-M := \{-m : m \in M\}$. For weight function w, let $-w$ denote the weight function obtained from w by pointwise multiplying its values by -1. For $\Pi \in \{\text{SAT}, \text{XSAT}, \text{NAESAT}\}$, let $T^{\Pi}(C)$ denote the set of all Π-models of C and similarly let $T^{\Pi}_{\mu}(C, w) \subseteq T^{\Pi}(C)$ denote the set of all μ-weight Π-models of (C, w), with $\mu \in \{\min, \max\}$.

3 Reduction Tools for Variable-Weighted Formulas

Let us collect some useful tools for later considerations. The first assertion slightly generalizes Lemma 7 in [16] restricted to weighted XSAT, and enabling us to reduce maximum weight problems to minimum weight problems in specific cases:

Lemma 1. *Let $\Pi \in \{\text{SAT}, \text{XSAT}, \text{NAESAT}\}$. If there exists an algorithm $\mathcal{A}$ solving* MINW-Π *for M-weighted members of a formula class $\mathcal{C} \subseteq$* CNF *in*

$O(f(\|C\|))$ time then $\mathcal{A}$ also solves MAXW-Π *for $(-M)$-weighted members of $\mathcal{C}$ and vice versa. Moreover, in case $M = -M$, algorithm $\mathcal{A}$ can easily be modified to $\mathcal{A}'$ solving* MAXW-Π *for M-weighted members of $\mathcal{C}$ in $O(f(\|C\|))$ time.*

Proof: Let $\mathcal{C} \subseteq$ CNF be a fixed formula class for which $\mathcal{A}$ is an algorithm solving MINW-Π for M-weighted input instances $C \in \mathcal{C}$. We claim that $T^{\Pi}_{\min}(C, w) = T^{\Pi}_{\max}(C, -w)$. From that claim the first assertion obviously follows. Moreover, if $M = -M$, given an M-weighted formula (C, w), let $\mathcal{A}'$ first compute $(C, -w)$ in linear time which then also is M-weighted, and then $\mathcal{A}'$ performs $\mathcal{A}$ on $(C, -w)$ finding an element $t \in T^{\Pi}_{\min}(C, -w)$, if existing, therefore $t \in T^{\Pi}_{\max}(C, w)$ as required. Since $\mathcal{A}$ at least must have linear running time, $\mathcal{A}'$ also has time bound $O(f(\|C\|))$.

To verify the claim let $t \in T^{\Pi}_{\min}(C, w)$, and assume $t \notin T^{\Pi}_{\max}(C, w')$ where $w' := -w$. Then there exists $t_0 \in T^{\Pi}(C)$ with $w'(t_0) > w'(t)$ which is equivalent to $-w'(t_0) < -w'(t)$ meaning $w(t_0) < w(t)$ contradicting $t \in T^{\Pi}_{\min}(C, w)$. Therefore $T^{\Pi}_{\min}(C, w) \subseteq T^{\Pi}_{\max}(C, -w)$. Analogously, we obtain $T^{\Pi}_{\max}(C, -w) \subseteq T^{\Pi}_{\min}(C, w)$.

The vice versa assertion stating that an algorithm solving MAXW-Π for $(-M)$-weighted members in $\mathcal{C}$ also solves MINW-Π for M-weighted formulas analogously follows from the claim that $T^{\Pi}_{\max}(C, w) = T^{\Pi}_{\min}(C, -w)$ holds, for an arbitrary $(-M)$-weighted formula (C, w), $C \in \mathcal{C}$. This claim is shown as the previous one. □

Next, we state a basic proposition relating bijections between Π-model spaces to bijections between weighted Π-model spaces, for $\Pi \in \{\text{SAT}, \text{XSAT}, \text{NAESAT}\}$. This result is a slight generalization of the same result restricted to weighted XSAT shown in [14,16]:

Proposition 1. *For arbitrary M-weighted formulas $(C, w), (C', w')$ $(M \subseteq \mathbb{R})$, assume that there exists a bijection*

$$F : T^{\Pi}(C) \ni t \mapsto t' := F(t) \in T^{\Pi}(C')$$

such that $(*)$: $w(t) = w'(t') + \alpha$, where $\alpha \in \mathbb{R}$ is a constant independent of t and t'. Then the restricted mapping $F_\mu := F|T^{\Pi}_{\mu}(C, w)$ is a bijection between $T^{\Pi}_{\mu}(C, w)$ and $T^{\Pi}_{\mu}(C', w')$, so we have $|T^{\Pi}_{\mu}(C, w)| = |T^{\Pi}_{\mu}(C', w')|$, for $\mu \in \{\min, \max\}$.

Proof: First consider the minimization case. Let $t \in T^{\Pi}_{\min}(C, w)$ and assume that $t' := F_{\min}(t) \notin T^{\Pi}_{\min}(C', w')$. Then there is a Π-model $t'_0 \in T^{\Pi}(C')$ with $w'(t'_0) < w'(t')$. Let $t_0 := F^{-1}(t'_0)$ be the corresponding Π-model of C. Applying $(*)$ twice we obtain $w(t_0) = w'(t'_0) + \alpha < w'(t') + \alpha = w(t)$, contradicting the assumption that t is minimum. Hence $F_{\min}(t) \in T^{\Pi}_{\min}(C', w')$ holds for each $t \in T^{\Pi}_{\min}(C, w)$.

Conversely, let $t' \in T^{\Pi}_{\min}(C', w')$ and assume $t := F^{-1}(t') \notin T^{\Pi}_{\min}(C, w)$. Then there is a Π-model $t_0 \in T^{\Pi}(C)$ with $w(t_0) < w(t)$. Let $t'_0 := F(t_0)$ be the corresponding Π-model of C'. As above, by $(*)$, we derive $w'(t'_0) = w(t_0) - \alpha < w(t) - \alpha = w'(t')$, contradicting the assumption that t' is minimum. Hence $F^{-1}(t') \in T^{\Pi}_{\min}(C, w)$ holds for each $t' \in T^{\Pi}_{\min}(C', w')$. Thus, F^{-1} restricted to $T^{\Pi}_{\min}(C', w')$ equals $F^{-1}_{\min}$ from which the assertion follows. Proving the assertion for maximum model spaces proceeds analogously. □

4 Optimum Weight 2-SAT

As is well known, 2-SAT, i.e., SAT restricted to 2-CNF can be decided and solved in linear time in the length of the formula [2]. However, a straightforward reduction from the minimum weight vertex cover problem (MINW-VC) in graphs tells us that the weighted version MINW-2SAT is NP-hard. To that end simply observe that the edges of the graph represent clauses of a monotone formula whose variables correspond to the graph vertices. Obviously a minimum weight model as defined above is equivalent to a minimum vertex cover, i.e., a smallest weight subset of vertices covering all graph edges. Therefore, MINW-VC and MINW-2SAT for monotone formulas are identical. In this section we provide an algorithm for optimum weight SAT restricted to arbitrarily variable-weighted 2-CNF (also called *quadratic*) formulas.

For Horn formulas, SAT can be decided and solved in linear time which is a well-known result [9,10]. The minimization problem for weighted Horn formulas can also be solved in linear time:

Lemma 2 ([18]). *Minimum weight satisfiability for a Horn formula H and weight function $w : V(H) \to \mathbb{R}_+$, can be solved in linear time.*

Observe that the maximization problem cannot be reduced to the minimization case as only non-negative weights are allowed.

If $P(C)$ is empty then C is a 2-CNF Horn formula, we thus obtain immediately:

Corollary 1. *For $C \in$ 2-CNF with $P(C) = \varnothing$,* MINW-XSAT *can be solved in linear time, where $w : V(C) \to \mathbb{R}_+$.* □

Only for *non-negatively* weighted 2-CNF formulas with n variables, MINW-SAT has been shown to be solvable in time $O(2^{0.5284 \cdot n})$ in [18,20] regarding it as a specific *mixed Horn formula* which can be represented as the union of a Horn and a quadratic formula:

Lemma 3 ([20]). *Minimum (resp. maximum) weight satisfiability can be solved in $O(2^{0.5284|V(C)|})$ time, for formulas $C \in$ 2-CNF and $w : V(C) \to \mathbb{R}_+$ (resp. $w : V(C) \to \mathbb{R}_-$).*

The proof uses the Johnson-Papadimitriou-Yannakakis algorithm [8] for generating all maximal independent sets in the graph with polynomial delay yielding time bound $O(2^{0.5284n})$, for n vertices.

The last result can be generalized to arbitrarily variable-weighted 2-CNF formulas resting on the next assertion stating that optimum weight Π-models are preserved in a certain sense when variables are complemented.

Lemma 4. *Let (C, w) with $C \in$ CNF, $w : V(C) \to \mathbb{R}$, and $X \subseteq V(C)$ be arbitrary. Then, for (C^X, w^X) with $w^X(x) := w(x)$, $\forall x \in V(C) \setminus X$, and $w^X(x) := -w(x), \forall x \in X$, we have for each fixed $\Pi \in \{\mathrm{SAT}, \mathrm{XSAT}, \mathrm{NAESAT}\}$:*
(i) $|T^\Pi(C)| = |T^\Pi(C^X)|$, given by $t \mapsto t^X$,
(ii) $|T^\Pi_\mu(C, w)| = |T^\Pi_\mu(C^X, w^X)|$, $\mu \in \{\min, \max\}$.

Proof: Let $\Pi \in \{\mathrm{SAT}, \mathrm{XSAT}, \mathrm{NAESAT}\}$ then t obviously is a Π-model of C iff t^X is a Π-model of C^X, for each $X \subseteq V(C)$. Indeed, let ℓ be a literal at any fixed position p in C regarded as a vector of length $\|C\|$, then the truth value at position p is $t(\ell)$. Now either holds $V(\ell) \in X$ or $V(\ell) \notin X$. In the first case $\bar{\ell}$ is the corresponding literal at position p in C^X having truth value $t^X(\bar{\ell})) = 1 - t(\bar{\ell}) = t(\ell)$. In the remaining case the truth value of ℓ is not affected. Therefore the truth value vectors of C, t, resp. C^X, t^X, are identical completing the argumentation and implying that $F : T^{\Pi}(C, w) \ni t \mapsto t^X \in T^{\Pi}(C^X, w^X)$ is a bijection of Π-model spaces, hence (i) is true.

Regarding (ii), first observe that $t^X(x) = 1 - t(x)$, for each $x \in X$, and $t^X(y) = t(y)$, for each $y \in V(C) - X$. Therefore, we have

$$w^X(t^X) = \sum_{x \in X} (-w(x))(1 - t(x)) + \sum_{x \in V(C) - X} w(x)t(x) = -w(X) + w(t)$$

Hence $w^X(t^X) = w(t) + \alpha$, where $\alpha := -w(X)$ is a constant as X is fixed. So, assertion (ii) can immediately be derived from Prop. 1. □

As already mentioned, the last observation helps us to solve minimum weight 2-SAT for arbitrarily weighted formulas:

Theorem 1. *Minimum (resp. maximum) weight satisfiability can be solved in* $O(2^{0.5284|V(C)|})$ *time, for formulas* $C \in$ 2-CNF *and* $w : V(C) \to \mathbb{R}$.

Proof: Let (C, w) with $C \in$ 2-CNF and $w : V(C) \to \mathbb{R}$ be an arbitrary input for MINW-2SAT. Define $X := \{x \in V(C) : w(x) < 0\} \subseteq V(C)$. Then compute a minimum weight model t of (C^X, w^X) due to Lemma 3 which is possible because $w^X : V(C^X) \to \mathbb{R}_+$.

According to Lemma 4, t^X then is a minimum model of (C, w). So the assertion for the minimization case follows because C^X, w^X, and t^X can be computed in linear time $O(|V(C)|)$ using appropriate data structures. The maximization case follows due to Lemma 1. □

If (C, w) is an $\mathbb{R}$-weighted formula possessing a set $X \subseteq V(C)$ such that C^X is a Horn formula, and only the variables in X are negatively weighted, then by Lemma 1 we can solve minimum weight SAT for (C^X, w^X) in linear time: By the last theorem, we then also obtain a solution for the original input C with $w : V(C) \to \mathbb{R}$, because we only have to complement the model values for the variables in X.

It should be noted that, via the above approach, we cannot improve on Theorem 9 in [20], i.e., we cannot show that minimum (resp. maximum) weight satisfiability for each mixed Horn formula C with arbitrary weights can be solved in $O(2^{0.5284|V(C)|})$ time. The reason is that a mixed Horn formula does not remain mixed Horn if a certain subset of variables is complemented. But fortunately 2-CNF formulas remain stable under complementations of arbitrary variable sets.

XSAT resp. NAESAT are the same for 2-CNF formulas containing no unit clauses: Formulas containing unit clauses obviously do not belong to NAESAT. However unit clauses are not critical as the corresponding literals must be set

to true regardless of variable weights. The optimization versions minimum and maximum weight XSAT resp. NAESAT are solvable in linear time for 2-CNF, cf. [13], Thm. 1.

5 The Weighted Dual Class $\mathrm{CNF}_+(\leq 2)$

In this section MINW-Π, resp., MAXW-Π for the case of monotone weighted input formulas (C, w), i.e. $C \in \mathrm{CNF}_+(\leq 2)$, $w : V(C) \to \mathbb{R}$, is treated, for each fixed $\Pi = \{\mathrm{XSAT}, \mathrm{SAT}, \mathrm{NAESAT}\}$. Recall that, by definition, each variable occurs in at most two distinct clauses of C, hence $|C(x)| \leq 2$, for each $x \in V(C)$. Observe that $\mathrm{CNF}(\leq 2)$ can be regarded as dual to 2-CNF in the sense that assigning a set S_x to each variable $x \in V(C)$ defined by $S_x := \{c \in C : x \in V(c)\}$ yields "variable-clauses" of length at most 2: $|S_x| \leq 2$.

The algorithmic strategy is as follows focusing first on MINW-Π, for each fixed $\Pi \in \{\mathrm{SAT}, \mathrm{XSAT}, \mathrm{NAESAT}\}$: We reduce MINW-$\Pi$ in polynomial time to a corresponding equivalent problem on an edge weighted graph that is closely related to the (edge-weighted) intersection graph of (C, w). Then we construct a polynomial time algorithm solving this graph problem, thereby yielding a minimum Π-model of (C, w) or responding that none such exists. An algorithm for the maximization version then is provided via Lemma 1.

Recall that the intersection graph of a monotone formula $C \in \mathrm{CNF}_+$ has a vertex for each clause, and two vertices are joined by an edge iff the corresponding clauses have non-empty intersection. We will make use of a modification of the intersection graph that we call the *clause graph* G_C associated to a variable-weighted $C \in \mathrm{CNF}_+(\leq 2)$ also incorporating variables that uniquely occur in C, i.e., in only one clause: Thus, if C admits no unique variables the clause graph simply is the intersection graph, such that each edge is labeled by a variable of least weight in the intersection and is weighted by that weight.

In case that C admits unique variables, make a copy of its intersection graph, pose to each edge in either copy its label and its weight as mentioned above. Finally, join each pair of vertices in either copy that correspond to the same clause by an edge iff the clause contains a unique variable; label that edge with a unique variable of least weight, and assign to it that weight value yielding G_C.

The clause graph has at most $2|C|$ vertices and $|V(C)|$ edges, and can obviously be built in $O(|C|^2 \cdot |V(C)|)$ time. Since each variable occurs in at most two clauses, i.e. $\forall x \in V(C) : |C(x)| \leq 2$, we have $|C| \leq \|C\| = \sum_{x \in V(C)} |C(x)| \leq 2|V(C)|$. If G_C is not connected we first compute its components in linear time then treat each component independently decreasing running times obviously.

For the case $\Pi = \mathrm{XSAT}$, we have:

Lemma 5 ([13]). *A minimum, resp. maximum,* XSAT*-model of* (C, w) *with* $C \in \mathrm{CNF}_+(\leq 2)$, $w : V(C) \to \mathbb{R}$, *can be computed, respectively, it can be reported that none exists in time* $O(|V(C)|^3)$.

The proof is based on the fact that a minimum weight perfect matching in the clause graph is equivalent to a minimum weight XSAT-model of (C, w), if exactly

the variables that label the selected matching edges are set to true. The running time is determined by the matching algorithm. We even have:

Theorem 2. *A minimum, respectively, maximum weight* XSAT*-model of* (C,w)*, with* $C \in \mathrm{CNF}(\leq 2)$*,* $w : V(C) \to \mathbb{R}$*, can be computed, respectively, it can be reported that none exists in* $O(|V(C)|^3)$ *time.* □

The proof is based on a polynomial time reduction transforming (C, w) to a weighted monotone formula (C', w') such that the induced mapping $F_{\min}^{\mathrm{XSAT}} : T_{\min}^{\mathrm{XSAT}}(C, w) \to T_{\min}^{\mathrm{XSAT}}(C', w')$ on the minimum XSAT-model spaces is a bijection. Then a minimum XSAT-solution t_0' of (C', w') can be transformed into a minimum XSAT-solution of (C, w) via $t_0 := {F_{\min}^{\mathrm{XSAT}}}^{-1}(t_0')$. Hence, we arrive at a polynomial time algorithm for MINW-XSAT on arbitrarily variable-weighted members in CNF(≤ 2). From the latter algorithm we obtain a polynomial time algorithm solving MAXW-XSAT according to Lemma 1.

The next results provide elementary transformation steps for eliminating pure negative literals, complemented pairs in clauses, and negative literals that have its positive complement in a different clause. Regarding pure literals we are done by Lemma 4, for $X = \{x\}$:

Corollary 2. *For* $\Pi \in \{\mathrm{SAT}, \mathrm{XSAT}, \mathrm{NAESAT}\}$*, and* (C, w)*, with* $C \in \mathrm{CNF}$*,* $w : V(C) \to \mathbb{R}$*, let* $x \in V(C)$ *be a variable only occuring negated in* C*. Then for* (C^x, w^x)*, where* $w^x : V(C) \to \mathbb{R}$ *is defined as* w *except for* $w^x(x) := -w(x)$*, we have* $|T_\mu^\Pi(C, w)| = |T_\mu^\Pi(C_x, w_x)|$*,* $\mu \in \{\min, \max\}$*.*

Complemented pairs in the same clause can be treated due to:

Lemma 6 ([14,16]). *For* $C \in \mathrm{CNF}$ *with* $w : V(C) \to \mathbb{R}$*, holds* $T^{\mathrm{XSAT}} = \varnothing$ *if there is* $c \in C$ *containing more than one complemented pairs. Let* $c \in C$ *contain exactly one complemented pair* $x, \overline{x}$*, and let* C_c *be the formula obtained from* C *by removing* c *and assigning all literals to 0 that occur in* $c' := c - \{x, \overline{x}\}$ *(which can be empty), and by finally removing all duplicate clauses. Let* w_c *be the restriction of* w *to* $V(C_c) = V(C) - V(c')$*, then there is a bijection providing* $|T_\mu^{\mathrm{XSAT}}(C, w)| = |T_\mu^{\mathrm{XSAT}}(C_c, w_c)|$*,* $\mu \in \{\min, \max\}$*. Moreover the transformation* (C, w) *to* (C_c, w_c) *as well as the* XSAT*-model space bijection can be computed in polynomial time.* □

And finally, negative literals of complemented pairs occuring in distinct clauses can be eliminated via:

Lemma 7 ([14,16]). *Let* $C \in \mathrm{CNF}$*,* $w : V(C) \to \mathbb{R}$ *such that no clause contains a complemented pair. Let* $c_i = \{x\} \cup u, c_j = \{\overline{x}\} \cup v \in C$ *where* $x \in V(C)$ *and* u, v *are literal sets. Let* C_{ij} *be obtained from* C *as follows:*
(1) $C_{ij} := C - \{c_i, c_j\} \cup \{u \cup v\}$
(2) set all literals in $u \cap v$ *to 0, then remove all duplicate clauses from the resulting formula.*
Let $w_{ij} := V(C_{ij}) \to \mathbb{R}$ *be the following weight function: For each* $y \in V(C_{ij}) - V(u \oplus v)$*, set* $w_{ij}(y) := w(y)$*, and moreover, only in case that* $u \oplus v \neq \varnothing$*, define:*

(1') if $V_+(u \oplus v) \cap V_-(u \oplus v) = \{z\}$*, then set*

$$w_{ij}(y) := \begin{cases} w(y) \quad , & \text{if } y \in V(u \oplus v) - \{z\} \\ w(z) + w(x), & \text{if } z = y \text{ and } \overline{z} \in u, z \in v \\ w(z) - w(x), & \text{if } z = y \text{ and } z \in u, \overline{z} \in v \end{cases}$$

(2') if $V_+(u \oplus v) \cap V_-(u \oplus v) = \varnothing$*, then set*

$$w_{ij}(y) := \begin{cases} w(y) \quad , & \text{if } y \in V(v - u) \\ w(y) + w(x), & \text{if } y \in V_-(u - v) \\ w(y) - w(x), & \text{if } y \in V_+(u - v) \end{cases}$$

Then we have:
(i) $V(C_{ij}) = V(C) - [\{x\} \cup V(u \cap v)]$*,* $C_{ij} \in \mathrm{CNF}(\leq 2)$*, and* $|C_{ij}| \leq |C| - 1$*,*
(ii) $|T_\mu^{\mathrm{XSAT}}(C, w)| = |T_\mu^{\mathrm{XSAT}}(C_{ij}, w_{ij})|$*, for* $\mu \in \{\min, \max\}$*.*
Moreover the transformation (C, w) *to* (C_{ij}, w_{ij}) *as well as the* XSAT*-model space bijection can be computed in polynomial time.* □

Next, consider $\Pi = \mathrm{SAT}$: Since C is monotone we can set to true each variable that is weighted non-positively, and remove all clauses that are satisfied thereby. All variables that are removed from the formula and are not yet set have strictly positive weights and appear only in clauses already satisfied, therefore we can set them to false. Moreover if the remaining formula has clauses containing unique variables only, remove these clauses and set to true exactly one variable of least weight and to false all other variables. All steps above can obviously be done in linear time $O(\|C\|)$. It remains a monotone formula (C', w') such that each variable has strictly positive weight and occurs in at most two clauses, no clause exclusively has unique variables. Therefore the clause graph contains no isolated vertices.

Recall that an *edge cover* in a graph $G = (V, E)$ of no isolated vertices is a subset $F \subseteq E$ such that each vertex $x \in V$ is incident to at least one edge in F. It is not hard to see that a minimum weight edge cover F in the clause graph $G_{C'}$ yields a minimum weight SAT-model of (C', w') by setting exactly those variables to true that label the edges in F.

It is well known that the minimum cardinality of an edge cover in G is closely related to the matching number $\nu(G)$ of G, i.e., the cardinality of a maximum matching according to the relation $|F| = |V| - \nu(G)$. This is easy to see: all $2\nu(G)$ vertices in a maximum matching are covered already, all remaining vertices are independent from each other and only have neighbours in the covered set. So, for each remaining vertex we have to choose exactly one edge not contained in the matching.

There is no immediate connection as above between maximum weight matchings and minimum weight edge covers [7,22]. However, let (G, w) be a connected edge weighted graph with $w : E \to \mathbb{R}_+ - \{0\}$. Transforming the edge weights $w(e)$ to $\hat{w}(e)$, for each $e = x - y \in E$, according to

$$(*)\ \hat{w}(x-y) := -w(x-y)+\min\{w(x-z) : z \in N(x)\}+\min\{w(y-z) : z \in N(y)\}$$

where $N(v) \subset V$ denotes the set of all neighbours of $v \in V$ in G, yields weight function $\hat{w} : E \to \mathbb{R}$, cf. e.g. [4]. Now, perform a general maximum weight matching algorithm, for arbitrarily edge-weighted graphs, on $(G, \hat{w})$; and let $M \subseteq E$, such that $\hat{w}(M) = \max\{\hat{w}(M') : M' \subset E \text{ is matching in } G\}$, be its result. Let $V(M) \subset V$ denote that part of the vertices in G incident to an edge in M.

Then we claim (i): for each $x \in V - V(M)$ there exists an edge $x - y$ with $y \in V(M)$, and (ii): selecting one such edge of least weight for each $x \in V - V(M)$ collected in M' provides a minimum weight edge cover $M \cup M'$ in (G, w). Observe that $\hat{w} = w$ in case w is a constant function, and by the procedure described we obtain a minimum cardinality edge cover as explained above.

To verify the first claim, let $x \in V - V(M)$ be such that all its neighbours are members in $V - V(M)$ and let $E(x)$ denote the set of corresponding edges. Let $e = x - y \in E(x)$ such that $w(e) = \min\{w(e') : e' \in E(x)\}$. Therefore, with $(*)$

$$\hat{w}(e) := -w(e) + w(e) + \min\{w(y - z) : z \in N(y)\} > 0$$

because at least $x \in N(y)$. Hence, e would enlarge $w(M)$ and must be contained in M yielding a contradiction.

For (ii), clearly an edge $e = x - y$ is included in M only if $\hat{w}(e) \geq 0$, hence in view of $(*)$ e is the edge of least weight covering x and y, as all w-values are strictly positive. Similary, for each $x \in V - V(M)$, by (i) there is an edge $x - y$ of least weight $w(x - y)$ and $y \in V(M)$. It remains to show that such an edge is a cover of least weight for x if there is also $e' = x - z$ with $y \in V - V(M)$ of least w-weight among all such egdes. Indeed, because e' is independent of M we have $\hat{w}(e') \leq 0$ therefore, by $(*)$, follows $w(e') > 0$. Thus, we have verified that $M \cup M'$ provides an edge of least weight for each $x \in V$ and has least possible cardinality, therefore it is a minimum weight edge cover in (G, w).

Observe that the transformation $w \to \hat{w}$ can be carried out in $O(|V| \cdot |E|)$ time. Moreover, it is well known that a maximum weight matching in G can be computed relying on Edmonds blossom algorithm for perfect weighted matchings in $O(|V|^2 \cdot |E|)$ time [4]. Hence, a minimum weight edge cover in (G, w) of strictly positive edge weights can be computed in $O(|V|^2 \cdot |E|)$ time.

Recalling that the clause graph has $O(|C|)$ vertices, $O(|V(C)|)$ edges, and $|C| \leq 2|V(C)|$, we obtain in summary:

Theorem 3. *A minimum, resp. maximum weight* SAT *model of* (C, w), $C \in \mathrm{CNF}_+(\leq 2)$, $w : V(C) \to \mathbb{R}$, *can be computed in time* $O(|V(C)|^3)$. □

Finally, we consider NAESAT, which for the unweighted case has been solved in [17] based on Euler tour techniques. Unfortunately, in the weighted case this approach does not apply.

Let $G = (V, E)$ be a connected graph, and for fixed $F \subseteq E$ let $F(x)$ denote the set of edges incident to $x \in V$. Now let $f, g : V \to \mathbb{Z}$ be two functions such that $f \leq g$. Recall that a *f-factor* in G is a set $M \subseteq E$ such that for each $x \in V$ holds

$|M(x)| = f(x)$, which in general does not exist. E.g., for $f = 1$, an f-factor is a perfect matching. More generally, an $[f, g]$-*factor* in G is an edge subset $M \subseteq E$ with $|M(x)| \in [f(x), g(x)]$ for each $x \in V$. For an edge-weighted graph (G, w) with $w : E \to \mathbb{R}$ an *optimum weight* $[f, g]$-factor, is an $[f, g]$-factor M of optimal weight $w(M)$.

Let (C, w) with $C \in \mathrm{CNF}_+(\leq 2)$ containing no unit clauses (otherwise the formula a priori admits no NAESAT-models) and $w : V(C) \to \mathbb{R}$. Each $c \in C$ containing exclusively unique variables can be minimally NAESAT-satisfied independently: If c has only variables of non-negative weights, then set exactly one of the smallest weight to 1 and all other variables to 0. If c has only variables of non-positive weights, then set exactly one of the greatest weight to 0 and all other variables to 1. Observe that also the case is included where all variables have weight 0, and we have the convention that the variable with the smallest (largest) index then, by definition, is that of smallest (largest) weight.

In all remaining cases c contains at least one strictly positive-weighted and at least one strictly negative-weighted variable, so we set all the latter to 1 and the remaining variables to 0. It is obvious that in this way we provided a minimal NAESAT-model for all clauses containing unique variables only, which therefore can be omitted from the formula.

Next consider a clause c in the remaining formula that contains more than one unique variable collected in $U \subset V(c)$. We intend to assign truth values to all except one of these unique variables such that the remaining fragment of the clause is indepent w.r.t. minimum weight NAESAT: If all variables in U have non-negative weights, set to 0 all except for exactly one of the smallest weight, which will not yet be assigned. If all variables in U have non-positive weights, set to 1 all except for exactly one of the greatest weight, which will not yet be assigned. In all remaining cases c has at least one variable of strictly positive weight and at least one of strictly negative weight. Set all the latter variables to 1, and set to 0 all of the remaining (non-negative) variables except for exactly one of the smallest weight, which will not yet be assigned. From each such clause c, a fragment c' remains containing only one unique variable.

Observe that the resulting formula (C', w'), with restricted weight function $w' := w|V(C')$, yields an edge-weighted clause graph $G_{C'}$ such that each vertex has at least degree 2. Moreover, each variable in (C', w') labels a unique edge in the clause graph, and vice versa. Finally, the earlier eliminated unique variables had, for each clause containing them, been set appropriately. Now it easily follows that a minimum weight NAESAT-model of (C', w') is provided by setting to 1 exactly the variables labeling the edges in a minimum weight $[1, \deg]$-factor, if existing, in $G_{C'}$. Here $\deg : V(G_{C'}) \to \mathbb{Z}$ denotes the degree function, i.e., $\deg(x) = |N(x)|$, for each $x \in V(G_{C'})$.

With standard linear programming techniques one can solve a related problem in polynomial time, namely the maximum weight $[f, g]$-matching problem [4]: Given an edge-weighted graph $(G = (V, E), w)$, and f, g as above, one searches for $\mu : E \to \mathbb{Z}$ such that for each $x \in V$ holds $\sum_{y \in N(x)} \mu(x - y) \in [f(x), g(x)]$ and $\sum_{e \in E} \mu(e) w(e)$ is maximal. The maximum weight $[f, g]$-factor

problem obviously gets the maximum weight $[f,g]$-factor problem if, one poses the further constraint $0 \leq \mu(e) \leq 1$ for each $e \in E$, then $\mu(E)$ is the characteristic vector of a matching. Hence the latter problem also can be solved in polynomial time. The minimization version can be derived easily from the maximization version in the same manner as described in Lemma 1. So we arrive at:

Theorem 4. *A minimum, resp. maximum,* NAESAT*-model of* (C,w) *with* $C \in \mathrm{CNF}_+(\leq 2)$*,* $w : V(C) \to \mathbb{R}$*, can be computed, respectively, it can be reported that none exists in polynomial time.* □

6 Concluding Remarks and Open Problems

We proved that minimum (and also maximum) weight SAT for 2-CNF formulas of n arbitrarily weighted variables can be solved in time $O(2^{0.5284n})$. So an open problem is to construct a faster algorithm for optimum weight 2-SAT. Clearly, for monotone formulas minimum weight 2-SAT is the same as minimum weight vertex cover. Thus the question arise whether one can provide an appropriate polynomial time monotonization scheme also reducing general minimum weight 2-SAT to minimum vertex cover such that the formula does not increase. However, the monotonization methods discussed in Section 5 unfortunately do not apply, because simple resolution fails.

Regarding the second part, we leave open the question whether there can be constructed monotonization schemes, as valid for XSAT [14,16], solving the optimization versions of SAT and NAESAT even for arbitrary, i.e., not necessarily monotone members of $\mathrm{CNF}(\leq 2)$. What is missing, is a weighted version of the simple resolution rule for the SAT and NAESAT cases (holding in the unweighted case [17]).

Indeed only simple resolution is missing, because pure literal elimination already is provided by Lemma 2. And regarding complemented pairs in clauses we have:

Proposition 2. *For* $\Pi \in \{\mathrm{SAT}, \mathrm{NAESAT}\}$ *and* $C \in \mathrm{CNF}$*,* $w : V(C) \to \mathbb{R}$*, assume there is* $c \in C$ *containing complemented pairs, where* $W := V(C) - V(C - \{c\})$*, i.e., the set of variables only occuring in* c *which may be empty. Let* $\hat{C}$ *be the relevant part of* C *obtained from* C *by setting to 0 all* $x \in W$ *with* $w(x) = 0$*, let* $\hat{w}$ *be the restriction of* w *to* $V(\hat{C})$*. Let* $\hat{C}_c := \hat{C} - \{c\}$ *and let* $\hat{w}_c$ *be the restriction of* $\hat{w}$ *to* $V(\hat{C}_c)$*, then we have* $T^{\Pi}_{\mu}(\hat{C},\hat{w}) \neq \varnothing$ *iff* $T^{\Pi}_{\mu}(C,w) \neq \varnothing$*, and* $T^{\Pi}_{\mu}(\hat{C},\hat{w}) \subseteq T^{\Pi}_{\mu}(C,w)$*; moreover, there is a bijection providing* $|T^{\Pi}_{\mu}(\hat{C},\hat{w})| = |T^{\Pi}_{\mu}(\hat{C}_c,\hat{w}_c)|$*,* $\mu \in \{\min,\max\}$*. Finally, the transformation from* (C,w) *to* $(\hat{C}_c,\hat{w}_c)$ *as well as the* Π*-model space bijection can be computed in polynomial time.*

Proof: We distinguish two cases. (1): $W = \varnothing$, i.e., each variable in $V(c)$ also occurs in C_c, then $C = \hat{C}$, $w = \hat{w}$, and obviously each $t \in T^{\Pi}(C)$ also is a member of $T^{\Pi}(C_c)$ and vice versa, for $\Pi \in \{\mathrm{SAT}, \mathrm{NAESAT}\}$ providing a bijection (namely the identity) between the Π model spaces. As $w = w_c$, in that case, we also have $(*)$ of Prop. 1; so we are done.

(2): $W \neq \varnothing$. Since c contains at least one complemented pair $x, \bar{x}$, c is Π-satisfied by any truth assignment. Therefore, and because the variables in W do not occur in $C - \{c\}$, we can set to 0 all $x \in W$ with $w(x) = 0$ implying $T_\mu^\Pi(\hat{C}, \hat{w}) \neq \varnothing$ iff $T_\mu^\Pi(C, w) \neq \varnothing$, and $T_\mu^\Pi(\hat{C}, \hat{w}) \subseteq T_\mu^\Pi(C, w)$. Thus it suffices to consider $(\hat{C}, \hat{w})$ in the following. If $\hat{W}$ denotes the set of all remaining variables in W we have $V(\hat{C}_c) = V(\hat{C}) - \hat{W}$, and claim that

$$F^\Pi : T_{\min}^\Pi(\hat{C}, \hat{w}) \ni t \mapsto F^\Pi(t) := t | V(\hat{C}_c) \in T_{\min}^\Pi(\hat{C}_c, \hat{w}_c)$$

is a bijection if the reverse is defined as the extension of $t' \in T_\mu^\Pi(C_c, w_c)$ to $V(\hat{C})$ by assigning to 1 exactly all variables in $\hat{W}$ with $w(x) < 0$ and the others to 1, hence $\hat{w}(\hat{W}) = w(W)$ is minimal. It is easy to see that F^Π is one-to-one, and indeed is a bijection of minimum Π-model spaces. The maximization case proceeds analogously. The running time assertions are obvious. □

References

1. Aho, A.V., Ganapathi, M., Tjiang, S.W.: Code Generation Using Tree Matching and Dynamic Programming. ACM Trans. Programming Languages and Systems **11** (1989) 491-516
2. Aspvall, B., Plass, M.R., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified Boolean formulas. Inform. Process. Lett. **8** (1979) 121-123
3. Cook, S.: The Complexity of Theorem Proving Procedures. In: Proceedings of STOC 1971. ACM (1971) 151-158
4. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. Wiley, New York (1998)
5. Fürer, M., Kasiviswanathan, S.P.: Algorithms for Counting 2-SAT Solutions and Colorings with Applications. ECCC Report No. 33 (2005)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, San Francisco (1979)
7. Gibbons, A.: Algorithmic Graph Theory. Cambridge University Press, Cambridge (1985)
8. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On Generating All Maximal Independent Sets. Inform. Process. Lett. **27** (1988) 119-123
9. Lewis, H.R.: Renaming a Set of Clauses as a Horn Set. J. ACM **25** (1978) 134-135
10. Minoux, M.: LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and Computer Implementation. Inform. Process. Lett. **29** (1988) 1-12
11. Plagge, G.: Über Variablen-Gewichtete X3SAT Optimierungs-Probleme. Diploma Thesis, Univ. Köln (2006)
12. Plagge, G., Porschen, S.: Solving optimum variable-weight Exact 3-Satisfiability in time $O(2^{0.16254n})$. Techn. Report, Univ. Köln (2006), in preparation
13. Porschen, S.: On Some Weighted Satisfiability and Graph Problems. In: Vojtas, P., et al. (eds.): Proceedings of SOFSEM 2005. Lecture Notes in Computer Science, Vol. 3381. Springer-Verlag (2005) 278-287
14. Porschen, S.: Solving Minimum Weight Exact Satisfiability in Time $O(2^{0.2441n})$. In: Deng, X., Du, D. (eds.): Proceedings of ISAAC 2005. Lecture Notes in Computer Science, Vol. 3827. Springer-Verlag (2005) 654-664

15. Porschen, S.: Counting All Solutions of Minimum Weight Exact Satisfiability. In: Calamoneri, T., Finocchi, I., Italiano, G.F. (eds.): Proceedings of CIAC 2006. Lecture Notes in Computer Science, Vol. 3998. Springer-Verlag (2006) 50-59
16. Porschen, S.: On variable-weighted exact satisfiability problems. Techn. Report zaik2006-526, Univ. Köln (2006)
17. Porschen, S., Randerath, B., Speckenmeyer, E.: Linear Time Algorithms for some Not-All-Equal Satisfiability Problems. In: Giunchiglia, E., Tacchella, A. (eds.): Proceedings of SAT 2003. Lecture Notes in Computer Science, Vol. 2919. Springer-Verlag (2004) 172-187
18. Porschen, S., Speckenmeyer, E.: Satisfiability Problems for Mixed Horn Formulas. In: Kleine Büning, H., Zhao, X. (eds.): Proceedings of the Guangzhou Symposium on Satisfiability and its Applications (2004) 106-113
19. Porschen, S., Speckenmeyer, E.: Worst case bounds for some NP-complete modified Horn-SAT problems. In: Hoos, H.H., Mitchell, D. (eds.): Proceedings of SAT 2004. Lecture Notes in Computer Science, Vol. 3542. Springer-Verlag (2005) 251-262
20. Porschen, S., Speckenmeyer, E.: Satisfiability of Mixed Horn Formulas. To appear in Discrete Appl. Math. (http://dx.doi.org/10.1016/j.dam.2007.02.010)
21. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of STOC 1978. ACM (1978) 216-226
22. White, L.J.: A parametric study of matchings and coverings in weighted graphs. PhD Thesis, University of Michigan (1967)

On the Boolean Connectivity Problem for Horn Relations

Kazuhisa Makino[1], Suguru Tamaki[2], and Masaki Yamamoto[3]

[1] Graduate School of Information Science and Technology, University of Tokyo, Tokyo, 113-8656, Japan
makino@mist.i.u-tokyo.ac.jp
[2] Graduate School of Informatics, Kyoto University, Kyoto, 606-8501, Japan
tamak@kuis.kyoto-u.ac.jp
[3] Graduate School of Informatics, Kyoto University, Kyoto, 606-8501, Japan
masaki.yamamoto@kuis.kyoto-u.ac.jp

Abstract. Gopalan et al. studied in ICALP06 [17] connectivity properties of the solution-space of Boolean formulas, and investigated complexity issues on the connectivity problems in Schaefer's framework. A set S of logical relations is SCHAEFER if all relations in S are either bijunctive, Horn, dual Horn, or affine. They conjectured that the connectivity problem for SCHAEFER is in $\mathcal{P}$. We disprove their conjecture by showing that there exists a set S of Horn relations such that the connectivity problem for S is co$\mathcal{NP}$-complete. We also show that the connectivity problem for bijunctive relations can be solved in $O(\min\{n|\varphi|, T(n)\})$ time, where n denotes the number of variables, φ denotes the corresponding 2-CNF formula, and $T(n)$ denotes the time needed to compute the transitive closure of a directed graph of n vertices. Furthermore, we investigate a tractable aspect of Horn and dual Horn relations with respect to characteristic sets.

1 Introduction

The Boolean satisfiability problem (satisfiability problem for short) is one of the central problems in the computational complexity theory. Schaefer proposed in [30] a framework for expressing variants of the satisfiability problem, and showed a dichotomy theorem: the satisfiability problem for certain classes of Boolean formulas is in $\mathcal{P}$ while it is $\mathcal{NP}$-complete for all other classes. From this theorem, we have that 2-SAT and HORN-SAT are in $\mathcal{P}$, while k-SAT for $k \geq 3$, NAE-SAT (or NOT-ALL-EQUAL SAT), and XSAT (or EXACT SAT) are all $\mathcal{NP}$-complete[1]. Since then, dichotomies or trichotomies have been established for several aspects of the satisfiability problem such as optimization [6,8,27], counting [7], inverse satisfiability [26], minimal satisfiability [19], unique satisfiability [18], 3-valued satisfiability [4] and propositional abduction [9].

[1] There are many classes of formulas that do not fit Schaefer's framework, but can be solved in polynomial time. Such examples include renamable Horn [28], extended Horn [5], and q-Horn [3], for example.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 187–200, 2007.

Very recently, Gopalan et al. studied in [17] connectivity properties of the solution-space of Boolean formulas, and investigated complexity issues on connectivity problems in Schaefer's framework [30], where the connectivity properties of disjunctive normal forms (DNFs) were studied by Ekin et al. [14]. The connectivity problem (CONN) is to decide if the solutions of a given Boolean formula φ on n variables induce a connected subgraph of the n-dimensional hypercube, while the st-connectivity problem (ST-CONN) is to decide if two specific solutions s and t of φ are connected. As mentioned in [17], connectivity properties of Boolean satisfiability merit study in their own right, since they shed light on the structure of the solution-space, and moreover, structural studies on the solution-space are important to analyze the satisfiability problem and algorithms for it [2]. They [17] established a dichotomy for the st-connectivity problem: It is polynomially solvable if given Boolean relations are *tight*, while it is $\mathcal{PSPACE}$-complete in all other cases. This reveals that the tractable side is larger than the one for the satisfiability problem. Namely, the tight class properly contains SCHAEFER that consists of the classes of bijunctive, Horn, dual Horn, and affine relations. For the connectivity problem, they established a dichotomy with the same boundary: One side is in co$\mathcal{NP}$ and the other side is $\mathcal{PSPACE}$-complete. Furthermore, they showed that the connectivity problem for the class of non-SCHAEFER and tight is co$\mathcal{NP}$-complete. However, they did not give us a complete picture of the complexity status of the connectivity, and conjectured that the connectivity problem for SCHAEFER is in $\mathcal{P}$ [2].

In this paper, we disprove their conjecture by showing that there exists a set S of Horn relations such that the connectivity problem for S is co$\mathcal{NP}$-complete. Notice that this does not mean that the connectivity problem for any set of Horn relations is intractable. We also give proofs that show it is tractable for bijunctive and affine relations. In particular, we show that the connectivity problem for bijunctive relations can be solved in $O(\min\{n|\varphi|, T(n)\})$ time, where n denotes the number of variables, φ denotes the corresponding 2-CNF formula, and $T(n)$ denotes the time needed to compute the transitive closure of a directed graph of n vertices [3]. It is known [31] that $T(n) = O(n^\omega)$, where $\omega \leq 2.376$.

We also investigate a tractable aspect of the intractable side (i.e., Horn and dual Horn relations). We consider the *semantic* (i.e., model-based) representation of Horn relations, instead of the traditional *syntactic* (i.e., formula-based) one. The model-based representation has been proposed as an alternative form of representing and accessing a logical knowledge base, e.g., [10,11,12,20,21,22,24,25]. In contrast to the formula-based representation, if we have the model-based representation, that is, if we are given the *characteristic set* of Horn relations, the connectivity problem is solvable in polynomial time. This strengthens the result in [14] that the connectivity problem for DNF formulas can be solved in

[2] Actually, they mentioned (without proofs) that the connectivity problem can be solved in polynomial time for bijunctive and affine relations. Thus, what remains is to show the exact complexity of the connectivity problem for Horn and dual Horn relations.

[3] We again note that the polynomiality is only mentioned in [17].

polynomial time, since model-based representation M is more compact than DNF representation. More precisely, for any DNF formula ψ, we have $|M| \leq n|\psi|$, where $|M| \ll |\psi|$ is expected in most cases.

The rest of the paper is organized as follows. In the next section, we review the basic Boolean concepts and fix notations. Section 3 presents a polynomial-time algorithm for bijunctive relations, and shows a proof of co$\mathcal{NP}$-completeness for a set of Horn relations. Section 4 considers the connectivity problem for model-based representation of Horn relations.

2 Preliminaries

We review the basic concepts of the classification of Boolean constraint satisfaction problems, which were introduced by Schaefer [30]. A *logical relation* R over k Boolean variables, which is called a *k-arity relation*, is a mapping from $\{0,1\}^k$ to $\{0,1\}$. We say that a k-arity relation R is *satisfied* by an assignment $t \in \{0,1\}^k$ if and only if $R(t) = 1$. Let S be a (finite) set of relations, and X be a set of Boolean variables. An *S-constraint* over X is defined as the form of $R(y_1, \cdots, y_k)$ for some k-arity relation $R \in S$ and some $\{y_1, \cdots, y_k\} \subseteq X$. We say that a collection φ of S-constraints over n variables is *satisfied* by an assignment $t \in \{0,1\}^n$, denoted by $\varphi(t) = 1$, if every S-constraint of φ is satisfied by t. We call such an assignment t a *satisfying assignment* or a *solution* for φ. In this framework, the satisfiability problem $\textsc{Sat}(S)$ is to decide if there exists a solution for a given collection φ of S-constraints. In this framework, several problems have been investigated. In this paper, we consider the connectivity problem, denoted by $\textsc{Conn}(S)$, which was introduced by [17].

Let H_n be the n-dimensional hypercube. Given a collection φ of S-constraints over n variables, we denote by $G(\varphi) = (V_\varphi, E_\varphi)$ the subgraph of H_n induced by the solutions of φ, that is, $V_\varphi = \{t \in \{0,1\}^n : \varphi(t) = 1\}$, and $(t, t') \in E_\varphi$ for $t, t' \in V$ if and only if the Hamming distance $d(t, t')$ between t and t' is one. The connectivity problem $\textsc{Conn}(S)$ is to decide if $G(\varphi)$ is connected for a given collection φ of S-constraints. In this paper, we assume that readers are familiar with the standard notions and notations of graph theory such as path, cycle and connected component.

Let X be a set of Boolean variables. A *literal* is a variable $x \in X$ or its negation $\overline{x}$, which are respectively called *positive* and *negative*. A *clause* is a disjunction of literals, whose *length* is defined as the number of literals in it. A clause is called *unit* if its length is one. A formula is called *conjunctive normal form* (*CNF*) if it is a conjunction of clauses. A CNF formula is called *2-CNF* if each clause is of length at most two, and *Horn* (resp., *dual Horn*) if each clause has at most one positive (resp., negative) literal. Given a formula φ over X, a set of $\{y_1, \cdots, y_k\} \subseteq X$, and $a_1, \cdots, a_k \in \{0,1\}$, we denote by $\varphi|_{y_1=a_1,\cdots,y_k=a_k}$ the formula obtained from φ by assigning y_i to a_i for $i = 1, \cdots, k$.

In this paper, we are interested in the connectivity problem $\textsc{Conn}(S)$ with respect to the following four types of relations.

Definition 1. Let R be a relation. We say that R is (1) *bijunctive* if it is expressible as a 2-CNF formula, (2) *Horn* if it is expressible as a Horn formula, (3) *dual Horn* if it is expressible as a dual Horn formula, and (4) *affine* if it is expressible as a system of linear equations over $\mathbb{GF}(2)$.

Definition 2. A set S of relations is SCHAEFER if at least one of the following holds: (1) Every relation in S is bijunctive; (2) Every relation in S is Horn; (3) Every relation in S is dual Horn; (4) Every relation in S is affine.

Since we are concerned with only these four types of relations, we simply deal with CNF formulas and systems of linear equations over $\mathbb{GF}(2)$, instead of relations representing them.

Given two assignments $t, t' \in \{0,1\}^n$, we define a *coordinate-wise partial order* $\leq$ as follows: $t \leq t'$ if $t_i \leq t'_i$ for all $1 \leq i \leq n$. Given a formula φ, we say that a satisfying assignment t is *locally minimal* for φ if t has no satisfying neighbor t' with $t' \leq t$, i.e., $\varphi(t') = 1$, $d(t', t) = 1$ and $t' \leq t$. Observe that t is locally minimal for φ if and only if, for each i with $t_i = 1$, there exists a clause C in φ that is falsified by flipping the value of t_i from t. In the latter case, we say that t satisfies the *locally minimal condition*. A path $P = t^{(0)}, t^{(1)}, \cdots, t^{(k)}$ in $G(\varphi)$ is called *monotone* (*decreasing*) if $t^{(i-1)} \geq t^{(i)}$ for all $i = 1, \cdots, k$.

For the connectivity of Horn formulas, the following characterization is known.

Lemma 1 (Gopalan et al. [17]). *Let φ be a Horn formula. Then, every component of $G(\varphi)$ contains a unique locally minimal assignment. Moreover, every satisfying assignment is connected to the locally minimal solution in the same component by a monotone path.*

We make use of the following lemma, which is easily derived from the above.

Lemma 2. *Let φ be a Horn formula without unit clauses over n variables (i.e., a Horn formula such that $\varphi(0^n) = 1$). Then, $G(\varphi)$ is connected if and only if there exists no locally minimal assignment other than 0^n.*

3 Complexity of the Boolean Connectivity Problems Within Schaefer

In this section, we provide a polynomial-time algorithm for bijunctive relations, and a proof of coNP-completeness for a set of Horn relations.

3.1 Tractable Cases for CONN(S)

This subsection shows that CONN(S) is polynomially solvable, if S is either bijunctive or affine.

First, we briefly see the affine case. We assume without loss of generality that the underlying variables of a formula appear in the formula. Given an affine formula φ, we note that $d(t, t') \geq 2$ for every pair of (distinct) satisfying assignments t and t' for φ. From this observation, $G(\varphi)$ is connected if and only

if φ has at most one satisfying assignment. Thus, it suffices to check whether φ is satisfiable and whether φ is uniquely satisfiable, if so. Any affine formula φ with n variables and m clauses can be regarded as a linear system $Ax = \mathbf{b}$ over the finite field $\mathbb{GF}(2)$, where A is an $m \times n$ 0-1 matrix, and x and $\mathbf{b}$ are respectively the transposes of $(x_1, \cdots, x_n)$ and a vector in $\mathbb{GF}(2)^n$. We can easily see that φ is satisfiable if and only if $\mathrm{rank}(A) = \mathrm{rank}([A, \mathbf{b}])$, and φ is uniquely satisfiable if and only if $\mathrm{rank}(A) = \mathrm{rank}([A, \mathbf{b}]) = n$. Since we can obtain the rank of a matrix in polynomial time, we have the following result.

Theorem 1. *Let S be a set of affine relations. Then* $\textsc{Conn}(S)$ *is polynomially solvable.*

We next consider the bijunctive case, i.e., 2-CNF formulas. In what follows, we assume that a given 2-CNF formula is satisfiable, since otherwise, we can easily decide the connectivity.

We first note that we may assume that a given (not necessarily 2-CNF) formula has no unit clause.

Proposition 1. *Let φ be a formula over $\{x_1, \cdots, x_n\}$. For an i with $1 \leq i \leq n$ and an $a \in \{0, 1\}$, if $V_\varphi = V_{\varphi|_{x_i=a}} \times \{a\}$ (i.e., unit clause x_i is implied by φ if $a = 1$, and $\overline{x}_i$ is implied by φ if $a = 0$), then $G(\varphi)$ is connected if and only if $G(\varphi|_{x_i=a})$ is connected.*

From this proposition, we deal with formulas without unit clauses: If a given formula φ contains a unit clause, say x_i, then we regard $\varphi|_{x_i=1}$ as an input formula. This can be applied until the resulting formula contains no unit clause. Note that this is possible in linear time.

We next note that a given 2-CNF formula may be assumed to be Horn.

Proposition 2. *Let φ be a formula over n variables. For an assignment $a \in \{0, 1\}^n$, let ψ be a formula obtained from φ by renaming a, i.e., $\psi(x) = \varphi(x \oplus a)$, where $\oplus$ denotes the component-wise exclusive-or. Then $G(\varphi)$ is connected if and only if $G(\psi)$ is connected.*

Proof. It follows from the fact that $d(t, t') = 1$ if and only if $d(t \oplus a, t' \oplus a) = 1$ for any assignments t and t'. □

From this proposition, we deal with Horn 2-CNF formulas: If a given 2-CNF formula φ is not Horn, then we construct a Horn formula $\psi(x) = \varphi(x \oplus a)$ by computing a satisfying assignment a for φ. Since $\psi(0^n) = 1$, we can see that ψ is Horn, and it can be computed in linear time [1].

In what follows, we assume that a given 2-CNF formula φ is satisfiable and Horn without unit clauses, which is equivalent to the condition that φ is satisfied by 0^n. We now present a notion of *core set*, followed by our key lemma.

Definition 3. Let φ be 2-CNF formula over X. We say that a subset $Y = \{y_1, \cdots, y_k\}$ of X is a *core set* for φ, if $k \geq 2$ and φ contains clauses $y_1 \vee \overline{y}_2, y_2 \vee \overline{y}_3, \cdots, y_{k-1} \vee \overline{y}_k, y_k \vee \overline{y}_1$ that form a cycle called a *core cycle* for φ. Furthermore, we say that a core set Y is *satisfiable* if $\varphi|_{y=1:y\in Y}$ is satisfiable.

Lemma 3. *Let φ be a 2-CNF formula, which is satisfiable and Horn without unit clauses. Then, $G(\varphi)$ is connected if and only if there exists no satisfiable core set for φ.*

Proof. Since $\varphi(0^n) = 1$ by assumption, we recall Lemma 2, i.e., $G(\varphi)$ is connected if and only if φ has no locally minimal non-zero assignment. We first show the only-if part. Let $X = \{x_1, \ldots, x_n\}$ be a variable set, and let $Y = \{x_1, \cdots, x_k\}$ be a satisfiable core set for φ, We assume that φ is satisfied by $t \in \{0,1\}^n$ such that $t_1 = 1, \cdots, t_k = 1, t_{k+1} = 1, \cdots, t_l = 1, t_{l+1} = 0, \cdots, t_n = 0$, where $2 \leq k \leq l \leq n$. Let t' be an assignment such that $t' \leq t$ and $t'_j = 0$ for exactly one j with $1 \leq j \leq k$. Since φ contains a core cycle, this means that t' does not satisfy φ. Hence we have no monotone path from t to 0^n, which proves the only-if part.

We next show the if part. Assume that $G(\varphi)$ is not connected. Then φ has a locally minimal non-zero assignment t, say, $t_1 = 1, \cdots, t_l = 1, t_{l+1} = 0, \cdots, t_n = 0$. Since φ is not satisfied by $t - e^{(j)}$, $j = 1, \cdots, l$, where $e^{(j)}$ is the j-th unit assignment, φ contains a $x_j \vee \overline{x}_{j'}$ with $1 \leq j' \leq l$. This implies that there exists a core cycle in these clauses. Hence there is a satisfiable core set Y for φ such that $Y \subseteq \{x_1 \cdots, x_l\}$, which completes the if part. □

By this lemma, it is not difficult to see that the connectivity problem for bijunctive relations is solvable in polynomial time.

Let φ be a 2-CNF formula φ, where we assume that it is satisfiable and Horn without unit clauses. We construct a directed graph $G = (V, E)$ from φ in the standard way [1]; i.e., $V = \{x, \overline{x} : x \in X\}$ and $E = \{(\overline{x}, y), (\overline{y}, x) : x \vee y \in \varphi\}$. As shown in [1], G represents implications for φ. Namely if G has a path from x to y, then $x = 1$ always implies $y = 1$, i.e., $\overline{x} \vee y$. Note that, by the assumption on φ, each strongly connected component consists of either only positive literals or only negative literals. Thus by the symmetricity of G (i.e., $(x, y) \in E$ if and only if $(\overline{y}, \overline{x}) \in E$), we write strongly connected components in G by $G_i = (V_i, E_i)$ and $G'_i = (V'_i, E'_i)$ for $1 \leq i \leq k$, where $V_i \subseteq X$ and $V'_i = \{\overline{x} : x \in V_i\}$. We note that any core cycle is contained in a single connected component G_i, and any connected component G_i with $|V_i| \geq 2$ contains a core cycle. Thus it suffices to check if, for each component $G_i = (V_i, E_i)$ with $|V_i| \geq 2$, $\varphi|_{x=1:x \in V_i}$ is satisfiable. This simply can be done as follows.

Let $H = (V_H = \{V_i, V'_i : 1 \leq i \leq k\}, E_H)$ be a directed graph obtained from G by identifying each connected component to a single vertex.

Lemma 4. *Let φ be a 2-CNF formula φ, which is satisfiable and Horn without unit clauses, and let H be defined as above. Then $G(\varphi)$ is connected if and only if, for every V_i with $|V_i| \geq 2$, there exists a path in H from V_i to V'_i.*

Proof. Let us first show the if part. Since H has a path from V_i to V'_i, $x = 1$, $x \in V_i$, always implies $\overline{x} = 1$. Thus we have no satisfiable core set for φ, which proves the if part by Lemma 3.

On the other hand, if $G(\varphi)$ is connected, we have no satisfiable core set for φ. Since any core set is contained in a connected component, say V_i, $\varphi|_{x=1:x \in V_i}$ is

unsatisfiable. This means that H contains two directed paths P_1 from V_i to V_j and P_2 from V_i to V_j' for some j. By the symmetricity of H, P_2 implies that H has a path P_3 from V_j to V_i'. Therefore, by concatenating P_1 and P_3, we have a path from V_i to V_i'. □

It follows from Lemma 4 that the connectivity problem for bijunctive relations can be solved in $O(n|\varphi|)$ time by checking the existence of n paths in H, where we note that H can be computed in linear time. If we first compute the transitive closure H^* of H, it can be computed in $\tilde{O}(n^\omega)$ time, where $\omega \leq 2.376$ and this is the current best bound for computing the transitive closure of a graph with n vertices.

Formally, our algorithm can be described in Figure 1.

`two-sat-conn`(φ) /* φ: a 2-CNF formula over X */

If φ is not satisfiable, then we output `YES` and halt.

Update φ to a Horn 2-CNF formula without unit clauses by using a satisfying assignment a for φ.

Construct a directed graph $G = (V, E)$ from φ by $V = \{x, \overline{x} : x \in X\}$ and $E = \{(\overline{x}, y), (\overline{y}, x) : x \vee y \in \varphi\}$.

/* Let $G_i = (V_i, E_i)$ and $G_i' = (V_i', E_i')$, $1 \leq i \leq k$, be the strongly connected components of G, where $V_i \subseteq X$ and $V_i' = \{\overline{x} : x \in V_i\}$. */

Construct H from G by identifying each connected component to a single vertex.

for each V_i of $V_1, \cdots, V_k$

if $|V_i| \geq 2$ and there is no path in H from V_i to V_i', then output `NO` and halt.

end-for-each

Output `YES`.

`end-of-two-sat-conn`

Fig. 1. An algorithm for bijunctive relations

Lemma 5. *Let φ be a 2-CNF formula of n variables. Then* `two-sat-conn`(φ) *correctly solves the connectivity problem for φ in $O(\min\{n|\varphi|, T(n)\})$ time, where $T(n)$ denotes the time needed to compute the transitive closure of a directed graph of n vertices.*

Therefore, we have the following positive result.

Theorem 2. *Let S be a set of bijunctive relations. Then* $\textsc{Conn}(S)$ *is polynomially solvable.*

3.2 Co$\mathcal{NP}$-Hardness for Relations in Horn and Dual Horn

In this subsection, we prove our main theorem.

Theorem 3. *Let S be a set of Horn (dual Horn) relations of arity 3. Then* $\textsc{Conn}(S)$ *is co$\mathcal{NP}$-complete.*

We only consider Horn relations S and show that the complement of $\textsc{Conn}(S)$ is NP-complete, since dual Horn relations are handled in a similar way.

First we give a necessary and sufficient condition for non-connectivity of Horn relations. In this section, we assume w.o.l.g. formulas contain no unit clause. By Lemma 2, a Horn formula is not connected if and only if there exists a locally minimal non-zero satisfying assignment, which can be represented by the following Boolean formula.

$$\Phi(\varphi) = \varphi \wedge \bigwedge_{x_i \in X} \left(\overline{x}_i \vee \Big(\bigvee_{C \in \varphi : P(C) = \{x_i\}} \bigwedge_{y \in N(C)} y \Big) \right). \tag{1}$$

Here, for a clause C, $P(C)$ and $N(C)$ respectively denote sets of variables that occur positively and negatively in C. Note that if $\{C \in \varphi : P(C) = \{x_i\}\}$ is empty, then $(\bigvee_{C \in \varphi : P(C) = \{x_i\}} \bigwedge_{y \in N(C)} y)$ is interpreted as false.

Lemma 6 (Logical formulation of non-connectivity). *Let φ be a Horn formula without unit clauses. Then there exists a locally minimal non-zero assignment of φ if and only if $\Phi(\varphi)$ is satisfied by a non-zero assignment.*

Proof. For the if part, let t be a non-zero satisfying assignment of $\Phi(\varphi)$. Note that t also satisfies φ. To confirm that this t satisfconnected.ies the locally minimal condition, pick an arbitrary variable x_i such that $t_i = 1$. Since t satisfies $\overline{x}_i \vee \bigvee_{C \in \varphi : P(C) = \{x_i\}} \left(\bigwedge_{y \in N(C)} y \right)$, φ contains a clause C such that t satisfies $\bigwedge_{y \in N(C)} y$, which implies that $t - e^{(i)}$ does not satisfy φ. Here $e^{(i)}$ denotes the i-th unit assignment. This completes the if part.

For the only-if part, let t be a locally minimal non-zero assignment of φ. For all $x_i \in X$, we show that t satisfies $\overline{x}_i \vee \bigvee_{C \in \varphi : P(C) = \{x_i\}} \left(\bigwedge_{y \in N(C)} y \right)$. It is obvious for x_i with $t_i = 0$. For x_i with $t_i = 1$, we have a clause C in φ such that $P(C) = \{x_i\}$ and $N(C) \subseteq \{x_j \in X : t_j = 1\}$. This proves the claim. □

By Lemmas 2 and 6, we have the following characterization.

Corollary 1 (Characterization of non-connectivity). *Let φ be a Horn formula without unit clauses. Then $G(\phi)$ is non-connected if and only if $\Phi(\varphi)$ is satisfied by a non-zero assignment.*

Now we are ready to prove the theorem.

The proof of Theorem 3. It follows from Proposition 1 and Corollary 1 that the complement of $\textsc{Conn}(S)$ for Horn relations S belongs to $\mathcal{NP}$. To show the $\mathcal{NP}$-hardness, we reduce to it 3-Uniform Hypergraph 2-Colorability, which is known to be $\mathcal{NP}$-complete (see SP4 in [16]). Let $\mathcal{H} = (V, \mathcal{E})$ be a 3-uniform hypergraph, where a hypergraph is called 3-*uniform* if $|E| = 3$ holds for all $E \in \mathcal{E}$. From $\mathcal{H}$ with $\mathcal{E} = \{E_1, \cdots, E_k\}$, we construct a 3-CNF Horn formula $\varphi_{\mathcal{H}}$ over a

variable set $X \cup X' \cup Y \cup Z \cup \{q\}$, where $X = \{x_v : v \in V\}$, $X' = \{x'_v : v \in V\}$, $Y = \{y_E : E \in \mathcal{E}\}$ and $Z = \{z_i : i = 1, 2, \ldots, k-2\}$, as follows:

$$\varphi_{\mathcal{H}} \equiv \bigwedge_{E \in \mathcal{E}} \left(\bigvee_{v \in E} \overline{x}_v \right) \wedge \bigwedge_{E \in \mathcal{E}} \left(\bigwedge_{v \in E} (y_E \vee \overline{x}_v) \right) \tag{2}$$

$$\wedge\, (\overline{y}_{E_1} \vee \overline{y}_{E_2} \vee z_1) \wedge \bigwedge_{i=1}^{k-3} (\overline{z}_i \vee \overline{y}_{E_{i+2}} \vee z_{i+1}) \wedge (\overline{z}_{k-2} \vee \overline{y}_{E_k} \vee q) \tag{3}$$

$$\wedge \bigwedge_{v \in V} \left((\overline{x}_v \vee x'_v \vee \overline{q})(x_v \vee \overline{x'_v} \vee \overline{q}) \right). \tag{4}$$

Example 1. Let $\mathcal{H} = (V, \mathcal{E})$ be a 3-uniform hypergraph defined by $V = \{1, 2, 3, 4\}$ and $\mathcal{E} = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}$. Then $\varphi_{\mathcal{H}}$ is given by

$$\begin{aligned}
\varphi_{\mathcal{H}} = {} & (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4)(\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_4)(\overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_4) \\
& \wedge (y_{E_1} \vee \overline{x}_1)(y_{E_1} \vee \overline{x}_2)(y_{E_1} \vee \overline{x}_3)(y_{E_2} \vee \overline{x}_1)(y_{E_2} \vee \overline{x}_2)(y_{E_2} \vee \overline{x}_4) \\
& \wedge (y_{E_3} \vee \overline{x}_1)(y_{E_3} \vee \overline{x}_3)(y_{E_3} \vee \overline{x}_4)(y_{E_4} \vee \overline{x}_2)(y_{E_4} \vee \overline{x}_3)(y_{E_4} \vee \overline{x}_4) \\
& \wedge (\overline{y}_{E_1} \vee \overline{y}_{E_2} \vee z_1)(\overline{z}_1 \vee \overline{y}_{E_3} \vee z_2)(\overline{z}_2 \vee \overline{y}_{E_4} \vee q) \\
& \wedge (\overline{x}_1 \vee x'_1 \vee \overline{q})(x_1 \vee \overline{x'_1} \vee \overline{q})(\overline{x}_2 \vee x'_2 \vee \overline{q})(x_2 \vee \overline{x'_2} \vee \overline{q}) \\
& \wedge (\overline{x}_3 \vee x'_3 \vee \overline{q})(x_3 \vee \overline{x'_3} \vee \overline{q})(\overline{x}_4 \vee x'_4 \vee \overline{q})(x_4 \vee \overline{x'_4} \vee \overline{q}).
\end{aligned}$$

Lemma 7. *Let $\mathcal{H}$ be a 3-uniform hypergraph and $\varphi_{\mathcal{H}}$ be a Horn formula constructed from $\mathcal{H}$ as above. Then $\mathcal{H}$ is 2-colorable if and only if $G(\varphi_{\mathcal{H}})$ is non-connected.*

Proof. Since $\varphi_{\mathcal{H}}$ constructed from a given $\mathcal{H}$ contains no unit clause by 3-uniformity of $\mathcal{H}$, we make use of the characterization of non-connectivity. The corresponding formula $\Phi(\varphi_{\mathcal{H}})$ in (1) can be written as follows:

$$\Phi(\varphi_{\mathcal{H}}) = \varphi_{\mathcal{H}} \wedge \bigwedge_{E \in \mathcal{E}} \left(\overline{y}_E \vee \bigvee_{v \in E} x_v \right) \tag{5}$$

$$\wedge\, (\overline{z}_1 \vee y_{E_1}\, y_{E_2}) \wedge \bigwedge_{i=1}^{k-3} (\overline{z}_{i+1} \vee y_{E_{i+2}}\, z_i) \wedge (\overline{q} \vee y_{E_k}\, z_{k-2}) \tag{6}$$

$$\wedge \bigwedge_{v \in V} \left((\overline{x}_v \vee x'_v q)(\overline{x'_v} \vee x_v q) \right). \tag{7}$$

Example 2. For a 3-uniform hypergraph $\mathcal{H}$ in Example 1, we have

$$\begin{aligned}
\Phi(\varphi_{\mathcal{H}}) = {} & \varphi_{\mathcal{H}} \wedge (\overline{y}_{E_1} \vee x_1 \vee x_2 \vee x_3)(\overline{y}_{E_2} \vee x_1 \vee x_2 \vee x_4) \\
& \wedge (\overline{y}_{E_3} \vee x_1 \vee x_3 \vee x_4)(\overline{y}_{E_4} \vee x_2 \vee x_3 \vee x_4) \\
& \wedge (\overline{z}_1 \vee y_{E_1} y_{E_2})(\overline{z}_2 \vee y_{E_3} z_1)(\overline{q} \vee y_{E_4} z_2)
\end{aligned}$$

$$\wedge\ (\overline{x}_1 \vee x'_1 q)(\overline{x'_1} \vee x_1 q)(\overline{x}_2 \vee x'_2 q)(\overline{x'_2} \vee x_2 q)$$
$$\wedge\ (\overline{x}_3 \vee x'_3 q)(\overline{x'_3} \vee x_3 q)(\overline{x}_4 \vee x'_4 q)(\overline{x'_4} \vee x_4 q).$$

To see the condition that $\Phi(\varphi_{\mathcal{H}})$ is satisfied by a non-zero assignment, we consider two cases depending on the value of q.

Fact 1. *$\Phi(\varphi_{\mathcal{H}})|_{q=0}$ is satisfied only if all the remaining variables are set to be 0.*

Proof. Setting $q = 0$ induces unit clauses $\overline{x}_v, \overline{x'_v}$ for every $v \in V$ by (7). This, together with (5) induces unit clause $\overline{y}_E$ for every $E \in \mathcal{E}$. Again, this, together with (6) induces unit clause $\overline{z}_i$ for every i. These imply Fact 1. □

Fact 2. *$\Phi(\varphi_{\mathcal{H}})|_{q=1}$ is satisfiable if and only if $\mathcal{H}$ is 2-colorable.*

Proof. Setting $q = 1$ induces $(\overline{z}_1 \vee y_{E_1}\, y_{E_2}) \wedge \bigwedge_{i=1}^{k-3}(\overline{z}_{i+1} \vee y_{E_{i+2}}\, z_i) \wedge (y_{E_k}\, z_{k-2})$ by (6), i.e., $y_{E_i} = z_i = 1$ for all i. Thus we can simplify $\Phi(\varphi_{\mathcal{H}})|_{q=1}$ as follows:

$$\Phi(\varphi_{\mathcal{H}})|_{q=1} = \bigwedge_{E\in\mathcal{E}} \left(\Big(\bigvee_{v\in E} \overline{x}_v \Big) \wedge \Big(\bigvee_{v\in E} x_v \Big) \right) \tag{8}$$
$$\wedge \bigwedge_{v\in V} \Big((\overline{x}_v \vee x'_v)(x_v \vee \overline{x'_v}) \Big). \tag{9}$$

Here we note that (8) is obtained from (2) and (5), while (9) is obtained from (7). It is easy to see that (8) is satisfiable if and only if $\mathcal{H}$ is 2-colorable. Since (9) just forces $x_v = x'_v$ for every $v \in V$, we have Fact 2. □

These facts, combined with Corollary 1, prove Lemma 7. □

This completes the proof of Theorem 3. □

4 Horn Relations Represented by Characteristic Sets

In this section, we investigate a tractable aspect of Horn and dual Horn relations. Specifically, we show that if we are given the characteristic set of Horn relations, the connectivity problem is solvable in polynomial time.

We recall that Horn relations have a well-known semantical characterization. Let for assignments $t, t' \in \{0,1\}^n$ denote $t \bigwedge t'$ their component-wise AND, and let for a set of assignment $M \subseteq \{0,1\}^n$ denote $Cl_\wedge(M)$ the closure of M under $\bigwedge$. Then, for every $M \subseteq \{0,1\}^n$, it holds that M is the set of satisfying assignments for some Horn formula φ (i.e, $M = V_\varphi$) if and only if $M = Cl_\wedge(M)$ [29] (see e.g., [10,24] for proofs). Namely, the set of satisfying assignments of a Horn formula is closed under the intersection $\bigwedge$, and any set of assignments which is closed under the intersection can be represented by a Horn formula. By this characterization, it is easy to see that any Horn formula has a unique minimal satisfying assignment. Here a satisfying assignment t for φ is called

minimal (resp., *maximal*) if there exists no other satisfying assignment t' for φ such that $t' \leq t$ (resp., $t' \geq t$). By definition, minimal assignments for φ are locally minimal, but not vice versa.

As discussed by Kautz *et al.* [20], a Horn formula φ is semantically represented by its characteristic assignment, where $v \in V_\varphi$ is called *characteristic* (or *extreme* [10]), if $v \notin Cl_\wedge(V_\varphi \setminus \{v\})$. The set of all characteristic assignments of φ, the *characteristic set of* φ, is denoted by $char(\varphi)$. Note that $char(\varphi)$ is unique and that $char(\varphi)$ contains all maximal satisfying assignments for φ.

Lemma 8. *For a Horn formula φ, let t^* be the unique minimal satisfying assignment for φ. Then $G(\varphi)$ is connected if and only if, for each maximal satisfying assignment t for φ, $G(\varphi)$ contains a monotone path between t^* and t.*

Proof. Since the only-if part is easily derived from Lemma 1, we only show the if part. We assume that, for each maximal assignment t of φ, $G(\varphi)$ contains a monotone path from t to t^*. We show that there is no locally minimal assignment other than t^*. This, together with Lemma 1 implies that $G(\varphi)$ is connected.

Let v be an arbitrary satisfying assignment for φ which is neither maximal nor minimal. Let t be a maximal assignment such that $t \geq v$. Since $G(\varphi)$ contains a monotone path from t to t^*, there exists an edge (u, w) in the path such that $u \geq v$ and $v \gneq (v \wedge w)$. Note that $v \wedge w$ is a satisfying assignment, and $d(v, v \wedge w) = 1$. This means that v is not locally minimal. □

By Lemma 8, the following simple algorithm checks the connectivity of Horn relations represented by the characteristic set.

Since $t \in V_\varphi$ if and only if $t = \bigwedge_{v \in char(\varphi): v \geq t} v$, for each assignment t in $char(\varphi)$ that includes maximal assignments for φ, the algorithm checks if $G(\varphi)$ contains a monotone path between t and t^*. Thus, from Lemma 8, algorithm `horn-sat-conn-from-charset`(φ) checks the connectivity of Horn functions.

`horn-sat-conn-from-charset`($char(\varphi)$)

/* $char(\varphi)$: the characteristic set of a Horn formula φ */

Let $t^* := \bigwedge_{t \in char(\varphi)} t$ and $M := char(\varphi)$

while ($M \neq \{t^*\}$)

Let t be an arbitrary element in $M \setminus \{t^*\}$

if there exists an index j s.t. $t_j = 1$ and $t - e^{(j)} = \bigwedge_{v \in char(\varphi): v \geq t - e^{(j)}} v$

then $M := (M \setminus \{t\}) \cup \{t - e^{(j)}\}$

Otherwise, output `NO` and halt

end-while

Output `YES`

`end-of-horn-sat-conn-from-charset`

Fig. 2. A naive algorithm for the Horn connectivity from the characteristic set

Theorem 4. *Given the characteristic set $char(\varphi)$ of a Horn formula φ, algorithm* `horn-sat-conn-from-charset`(φ) *checks its connectivity in $O(n^3|char(\varphi)|^2)$ time.*

Proof. Since the correctness of the algorithm follows from Lemma 8 and the discussion before the description, we only show its time complexity.

Clearly, we can initialize M and t^* in $O(n|char(\varphi)|)$ time. For each $t \in M \setminus \{t^*\}$, we can test if there exists an index j such that $t_j = 1$ and $t - e^{(j)} = \bigwedge_{w \in char(\varphi): w \geq t - e^{(j)}} w$ in $O(n^2|char(\varphi)|)$ time. Since we have at most $n|char(\varphi)|$ such t's, this requires $O(n^3|char(\varphi)|^2)$ time. Therefore, in total, the algorithm requires $O(n^3|char(\varphi)|^2)$ time. □

We now improve the complexity. For an assignment t, let $S_t = \{j \mid t_j = 0\}$. It

```
horn-sat-conn-from-charset2(char(φ))
  /* char(φ): the characteristic set of a Horn formula φ */
    Let t* := ⋀_{t∈char(φ)} t
    for each t of char(φ)
        Let S := {S_v | v ∈ char(φ)} and X := S_t
        while (∃S ∈ S with |S \ X| ≤ 1)
            Let S be an arbitrary element in S
            if |S \ X| = 0, then S := S \ {S}
            if |S \ X| = 1, then S := S \ {S} and X := X ∪ S
        end-while
        if X ≠ S_{t*}, then output NO and halt
    end-for-each
    Output YES
end-of-horn-sat-conn-from-charset2
```

Fig. 3. A faster algorithm for the Horn connectivity from the characteristic set

is not difficult to see that algorithm `horn-sat-conn-from-charset2`$(char(\varphi))$ checks the connectivity of a Horn formula φ: In the for-loop, we check if there exists a monotone path from each $t \in char(\varphi)$ to t^*. In the while-loop, we maintain a variable set X such that the corresponding assignment t^X (i.e., $t^X_j = 0$ if $j \in X$, and 1 otherwise) is reachable from t by a monotone path. Observe that t^X is not locally minimal if and only if there is a set S in the current $\mathcal{S}$ such that $|S \setminus X| = 1$. Moreover, the while-loop requires $O(n|char(\varphi)|)$ time, if $\mathcal{S}$ is stored in the proper data structure. Thus we have the following theorem.

Theorem 5. *Given the characteristic set $char(\varphi)$ of a Horn formula φ, algorithm* `horn-sat-conn-from-charset2`(φ) *checks its connectivity in $O(n|char(\varphi)|^2)$ time.*

Remark 1. This strengthens the result in [14] that the connectivity problem for DNF formulas can be solved in polynomial time, since the characteristic set $char(\varphi)$ is more compact than DNF representation ψ. More precisely, for any DNF formula ψ, we have $|char(\varphi)| \leq n|\psi|$, where $|char(\varphi)| \ll |\psi|$ is expected in most cases.

Remark 2. For Horn relations, formula-based (i.e., CNFs) and model-based (characteristic sets) representations are *orthogonal* in the sense that the one side may be exponentially larger than the other one. Therefore, the results in this section do not conflict with Theorem 3 in the previous section. We further remark that the transformation between a Horn formula φ and the characteristic set $char(\varphi)$ is at least as difficult as the monotone dualization problem [22,23], which is known to be solved in output quasi-polynomial time [13,15].

Acknowledgement

The authors thank anonymous referees for valuable comments.

References

1. B. Aspvall, M. F. Plass and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters* 8; pp. 121-123, 1979.
2. D. Achlioptas and F. Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. in *Proceeding of 38th ACM Symposium on Theory of Computing*, pp. 130–139, 2006.
3. E. Boros, T. Crama, P. L. Hammer, and M. E. Saks, A Complexity Index for Satisfiability Problems, SIAM J. Comput., 23(1), pp45-49, 1994.
4. A. Bulatov. A dichotomy theorem for constraints on a three-element set. in *Proceeding of 43rd IEEE Symposium on Foundations of Computer Science*, pp. 649–658, 2002.
5. V. Chandru and J. N. Hooker, Extended Horn sets in propositional logic, J. ACM, 38(1), pp205-221, 1991.
6. N. Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *Journal of Computer and System Sciences*, 51: pp. 511-522,1995.
7. N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125: pp. 1-12, 1996.
8. N. Creignou, S. Khanna, and M. Sudan. *Complexity classification of Boolean constraint satisfaction problems.* SIAM Monographs on Discrete Mathematics and Applications, 2001.
9. N. Creignou and B. Zanuttini. A complete classification of the complexity of propositional abduction. *SIAM Journal on Computing*, 36: pp. 207 - 229, 2006.
10. R. Dechter and J. Pearl. Structure identification in relational data. *Artificial Intelligence*, 58: pp. 237–270, 1992.
11. T. Eiter, T. Ibaraki and K. Makino. Computing intersections of Horn theories for reasoning with models. *Artificial Intelligence* 110: pp. 57-101, 1999.

12. T. Eiter and K. Makino. *Generating all abductive explanations for queries on propositional Horn theories*, KBS Research Report INFSYS RR-1843-03-09, Institute of Information Systems, Vienna University of Technology, 2006.
13. T. Eiter, K. Makino, and G. Gottlob. *Computational aspects of monotone dualization: A brief survey.* KBS Research Report INFSYS RR-1843-06-01, Institute of Information Systems, Vienna University of Technology, 2006.
14. O. Ekin, P. L. Hammer and A. Kogan. On connected Boolean functions. *Discrete Applied Mathematics*, 96–97: pp. 337–362, 1999.
15. M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21: pp. 618–628, 1996.
16. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1978.
17. P. Gopalan, P. G. Kolaitis, E. N. Maneva, and C. H. Papadimitriou. The connectivity of Boolean satisfiability: Computational and structural dichotomies. in *Proceeding of 33rd International Colloquium on Automata, Languages and Programming* (*ICALP'06*), pp. 346-357, 2006.
18. L. Juban. Dichotomy theorem for the generalize unique satisfiability problem. in *Proceedings of 12ht International Symposium of Fundamentals of Computation Theory*, LNCS 1684: pp. 327-337, 1999.
19. L. Kirousis and P. Kolaitis. The complexity of minimal satisfiability problems. *Information and Computation*, 187: pp. 20-39, 2003.
20. H. Kautz, M. Kearns, and B. Selman. Reasoning With characteristic models. In *Proceedings AAAI-93*, pp. 34–39, 1993.
21. H. Kautz, M. Kearns, and B. Selman. Horn approximations of empirical data. *Artificial Intelligence*, 74: pp. 129–245, 1995.
22. D. Kavvadias, C. Papadimitriou, and M. Sideri. On Horn envelopes and hypergraph transversals. In W. Ng, editor, *Proceedings 4th International Symposium on Algorithms and Computation (ISAAC-93)*, LNCS 762, pages 399–405, Hong Kong, December 1993.
23. R. Khardon. Translating between Horn representations and their characteristic models. *Journal of AI Research* 3: pp. 349-372, 1995.
24. R. Khardon and D. Roth. Reasoning with models. *Artificial Intelligence*, 87(1/2): pp. 187–213, 1996.
25. R. Khardon and D. Roth. Defaults and relevance in model-based reasoning. *Artificial Intelligence*, 97: pp. 169–193, 1997.
26. D. Kavvadias and M. Sideri. The inverse satisfiability problem. *SIAM Journal on Computing*, 28: pp. 152-163, 1998.
27. S. Khanna, M. Sudan, L. Trevisan, and D. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30: pp. 1863-1920, 2001.
28. H. R. Lewis, Renaming a set of clauses as a Horn set, J. ACM, 25, pp. 134-135, 1978.
29. J. McKinsey. The decision problem for some classes of sentences without quantifiers. *Journal of Symbolic Logic*, 8: pp. 61–76, 1943.
30. T. J. Schaefer. The complexity of satisfiability problems. in *Proceeding of 10th ACM Symposium of Theory of Computing*, pp. 216-226, 1978.
31. A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, 2003.

A First Step Towards a Unified Proof Checker for QBF

Toni Jussila[1], Armin Biere[1], Carsten Sinz[2], Daniel Kröning[3], and Christoph M. Wintersteiger[3]

[1] Formal Models and Verification, Johannes Kepler University, Linz
[2] Wilhelm-Schickard-Institute for Computer Science, University of Tübingen
[3] Computer Systems Institute, ETH Zürich

Abstract. Compared to SAT, there is no simple concept of what a solution to a QBF problem is. Furthermore, as the series of QBF evaluations shows, the QBF solvers that are available often disagree. Thus, proof generation for QBF seems to be even more important than for SAT. In this paper we propose a new uniform proof format, which captures refutations and witnesses for a variety of QBF solvers, and is based on a novel extended resolution rule for QBF. Our experiments show the flexibility of this new format. We also identify shortcomings of our format and conjecture that a purely resolution based proof calculus is not powerful enough to trace the most efficient solvers.

1 Introduction

The decision problem for the logic of Quantified Boolean Formulas (QBF) is the canonical PSPACE-complete problem. A vast number of problems can succinctly be formulated in QBF. Every finite two-player game can be modeled in QBF [1, 2]. A multitude of AI planning [3, 4, 5], and modal logic problems [6] can be formulated in QBF, but also unbounded and bounded model checking for finite-state systems [7, 8, 9], as well as other formal verification problems [10, 5].

There exist many different flavors of QBF solvers, based on DPLL [11, 12, 13, 14, 15, 16], Q–Resolution [17], BDDs [18], Skolemization [19] or Hyper Binary Resolution [20]. However, state-of-the-art QBF solvers are not yet reliable enough. The validity of many hard instances at recent QBF competitions had to be 'guessed' by means of a majority vote of the contestants, and often the solvers disagree [21].

Certificates for the decision problem of propositional logic (SAT) are easy to define. For satisfiable instances, solvers provide a satisfying assignment, and in case of an unsatisfiable instance, a resolution proof is computed. In both cases, the output can be verified easily by means of efficient (polynomial) and easy-to-inspect proof checkers. The satisfying assignment, or the resolution proof, correspond to a *certificate* of the correctness of the result. In case of SAT, these certificates serve many additional purposes: for example, satisfying assignments are often used as counterexamples. Resolution proofs are often used as an input

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 201–214, 2007.

for other algorithms. As an example, the computation of Craig interpolants relies on a resolution proof.

A certificate in the context of QBF refers to a *proof* of validity or invalidity of a formula. As in the case of unquantified SAT, certificates can serve dual purposes: first, they establish trust in the correctness of the result. The second motivation is to use certificates as input to other algorithms. When formulating a two–player game as a QBF, we wish to determine if there exists a winning strategy. If so, we are also interested in the strategy itself. Besides knowing that we *can* win the game, we also want to know *how* to win. The same holds for invalid formulas: not only do we want to know that a formula is invalid, we also want to find out *why* there is no solution and we wish to convince ourselves of that fact in a, preferably, concise way.

As shown by Tseitin [22], allowing definitions of fresh variables in Boolean formulas can exponentially shorten refutations. In this paper, we extend Tseitin's result to the quantified setting. We apply it in our proposal for a certificate format, which allows variable definitions of predefined structure to simplify the extension of QBF solvers with certificate generation code.

To the best of our knowledge, there exist only two suggestions for QBF certificates: One in the form of an "inference log", from which a BDD–based model or a refutation for a QBF can be reconstructed [23], and a method to generate unsatisfiable cores from traces of search–based solvers [24]. (In the QBF setting—similar to propositional logic—an unsatisfiable core is a subset of the clauses of a QBF formula in prenex normal form, which is still unsatisfiable.)

The first approach probably does not scale well because of the growth of the BDDs. For some instances it takes considerably more time to reconstruct the model from the inference log than it took to generate the model in the first place [23]. The inference log that is provided can serve as a refutation trace. However, due to five different inference strategies that the solver can choose, there are many different kinds of instructions in the inference logs. Among them are context switches between inference styles, explicit and symbolic variants of inference rules, such as resolution, substitution, and assignment, but also rollback and commit operations to undo earlier operations, as well as "other control information" [23]. This set of instructions, tailored to keep the overhead of the solver as small as possible, results in the need for a heavyweight proof checker.

The second approach provides only an unsatisfiable core and depends on the particular QBF solver used. In essence, both approaches just "trace", what the solver is doing and are thus far from a unified format that could be used to certify the computation of QBF solvers based on totally different algorithms.

However, our proposal is only a first step towards a uniform format. It clearly lacks the ability to capture important features of certain QBF solvers, such as *long distance resolution* [12], and *expansion* [17, 19]. We conjecture that the extension rule is not enough to linearly trace the proof process of such solvers.

Our extension rule is described in Section 2, and we discuss how to apply it to the major QBF solving algorithms. In order to evaluate the overhead of

generating certificates, we extend three different solvers with this capability. These implementations are described in Sec. 3. We provide the results of the evaluation in Sec. 4.

2 Theory

We consider closed Quantified Boolean Formulas (QBF) in prenex normal form. Thus, a formula is the concatenation of a quantifier prefix and a matrix of clauses. Let V be an infinite set of variables and let $\Omega\colon V \to \{\exists, \forall\}$ be a function that marks variables either as existential ($\exists$) or as universal ($\forall$). We define an order $<$ over V such that $x_1 < x_2$ for $x_1, x_2 \in V$ iff x_2 is in the scope of x_1, i.e., 'larger' variables are in the scope of 'smaller' variables.

The set of literals contains all variables and their negations $\neg v$ with $v \in V$. As usual, we extend the notion of negation to literals and identify $\neg\neg v$ with v. We also extend the order $<$ and Ω to literals in the natural way (without ordering the two literals of a variable). A *clause* is a disjunction of literals. A clause is trivial if it contains a literal and its negation. An empty clause is a clause without literals. A quantified formula in conjunctive normal form (CNF) is a conjunction of clauses. We assume the quantifier prefix Ω to be implicitly given, and do not mention it explicitly. In the following we just use the term formula to denote a quantified formula in CNF. A formula without clauses is called the empty formula.

A variable of a formula is called the innermost (resp. the outermost) variable if it is maximal (resp. minimal) among all variables of the formula with respect to the order $<$. The semantics $[\![f]\!]$ of a formula f is defined recursively by expanding the outermost variable x of a formula f as follows. If $\Omega(x) = \exists$ then define $[\![f]\!]$ as $[\![f\{x \leftarrow 0\}]\!] \vee [\![f\{x \leftarrow 1\}]\!]$, where the cofactor $f\{x \leftarrow c\}$ of f is f in which every occurrence of x is replaced by the Boolean constant c. More precisely, for $c = 0$, clauses containing $\neg x$ are deleted and x is removed from all clauses, and similarly for $c = 1$. If $\Omega(x) = \forall$, let $[\![f]\!] = [\![f\{x \leftarrow 0\}]\!] \wedge [\![f\{x \leftarrow 1\}]\!]$. Note that empty clauses (resp. formulas) are equivalent to the Boolean constant 0 (resp. 1) and are thus invalid (resp. valid).

Two non-trivial clauses C and D can be *resolved* iff C contains a literal l and D its negation $\neg l$. The result of a resolution is the *resolvent* clause, which is obtained as a disjunction of all literals in C and D except l and $\neg l$. This is the same definition as for unquantified formulas. If a clause C contains a universal literal l that is larger than all existential literals in C, then it can be removed from C. The process of removing literals according to this rule is called *forall-reduction* [25, 17] and the result obtained from repeated application until no more literals can be removed is called the *forall-reduct* of C. A *Q-resolution* step consists of a resolution step followed by forall-reduction of the resolvent.

The calculus described above is able to simulate the resolution-based, sound, and complete refutation calculus of [25]. This also allows tracing refutation proofs of QBF solvers based on DPLL [15] efficiently, even with further optimizations

such as trivial falsity [15], SAT and QBF based learning [5, 12, 16], and hyper binary resolution [20].

We conjecture that in order to trace other algorithms – for instance, algorithms that are structural or BDD-based – a much stronger proof system is necessary. In order to obtain such a stronger proof system, which is one of our main contributions in this paper, we add the following quantified extension rule. It is an adaptation of the classical extension rule [22] to the quantified setting.

Definition 1 (Quantified Extension Rule). *Let y be a fresh variable, which does not occur in formula f, and let g be a formula that may only contain y and variables in f. Furthermore, we demand that for any assignment to variables in f there exists an assignment to y that satisfies g. We also require $\Omega(y) = \exists$ and $z \leq y$ for all variables z in g. Then the* Quantified Extension Rule *extends f by g, i.e., it adds g conjunctively to f. We then call y a* defined variable *and g a* definition *for y with respect to f.*

Note that g does not need to enforce a functional dependency of y on other variables in g. As a relation, g has to be total, e.g., it cannot define a partial function, but it can be non-deterministic, e.g., the value of y does not need to be unique. This is a slight generalization of the classical extension rule for propositional logic [22].

In adapting the extension rule to the QBF setting, the crucial question is where to "put" new variables, e.g., how defined variables are ordered with respect to variables that already occur in the formula. It seems intuitive that new variables can only be existential. It is also clear that they cannot be moved further out than the innermost variable on which they depend. In the experimental section we show that we actually need this freedom to move defined variables as far out as possible. Keeping them in the innermost existential scope, as for instance in the Tseitin encoding of a non–CNF QBF formula, is insufficient.

To enforce that proofs are polynomially checkable, one can fall back to the original idea of Tseitin [22] and restrict the set of functions that can be defined and the way they are encoded in to clauses. This is what we suggest to use in practice. Useful functions are the constants, equality, negation, if-then-else and conjunction. Conjunction is sufficient:

Definition 2 (Restricted Quantified Extension Rule). *Let l, r be two literals over variables in the formula f, and y be a fresh variable, which does not occur in f. Let g be the conjunction of the following 3 clauses $(\overline{y} \vee l)$, $(\overline{y} \vee r)$, and $(y \vee \overline{l} \vee \overline{r})$. We also require $\Omega(y) = \exists$ and $z \leq y$ for all variables z in g. The formula f can be* extended *by adding g.*

The soundness of the restricted rule follows from the soundness of the generic rule, which is proved next. It turns out that this rule is enough to produce proofs from refutations of our BDD-based QBF solver EBDDRES in linear time. Refer to Sec. 4 for details.

For the proof of the following theorem we need the distributivity of substitution (resp. cofactoring) over Boolean operators, which we formulate for conjunctions as follows without proof:

Lemma 1. $(f \wedge g)\{x \leftarrow c\} \equiv f\{x \leftarrow c\} \wedge g\{x \leftarrow c\}$

Note that the post-fix operator for substitution has greater binding power than Boolean operators. The right hand side of the equivalence in the Lemma is thus read as $(f\{x \leftarrow c\}) \wedge (g\{x \leftarrow c\})$. In the following we will omit parenthesis as in Lemma 1 whenever possible. The next theorem shows that adding definitions is sound and does not change the semantics of a formula.

Theorem 1 (Soundness of Quantified Extension Rule). *Let g be a definition for y with respect to f. Then $[\![f]\!] = [\![f \wedge g]\!]$.*

Proof. The proof is by induction on the number of variables in $f \wedge g$. Let x be the outermost variable in $f \wedge g$. First assume that x is different from y, the variable defined by g, and $\Omega(x) = \exists$. The definitions and the lemma imply:

$$\begin{aligned}
[\![f \wedge g]\!] &= [\![(f \wedge g)\{x \leftarrow 0\}]\!] \vee [\![(f \wedge g)\{x \leftarrow 1\}]\!] \\
&= [\![f\{x \leftarrow 0\} \wedge g\{x \leftarrow 0\}]\!] \vee [\![f\{x \leftarrow 1\} \wedge g\{x \leftarrow 1\}]\!] \\
&= [\![f\{x \leftarrow 0\}]\!] \vee [\![f\{x \leftarrow 1\}]\!] \;=\; [\![f]\!]
\end{aligned}$$

In the next to last step we have to apply the induction hypothesis twice for the definitions $g\{x \leftarrow c\}$ with respect to $f\{x \leftarrow c\}$. The second case with $\Omega(x) = \forall$ is identical, except that all the disjunctions '$\vee$' are replaced by conjunctions '$\wedge$'.

In the base case we assume that $x = y$. From the definition of the extension rule, we know that $\Omega(x) = \exists$, and that x does not occur in f. Since all variables z in g have to be smaller or equal to y, g is either constant or only contains y as variable. Therefore g is either equivalent[1] to the Boolean constant 1, to the literal $\neg x$, or to the literal x. Otherwise, it is impossible to satisfy g by assigning a value to x. If $g \equiv 1$, then $[\![f \wedge g]\!] = [\![f \wedge 1]\!] = [\![f]\!]$. Without loss of generality, assume $g \equiv x$. Then

$$\begin{aligned}
[\![f \wedge g]\!] &= [\![(f \wedge g)\{x \leftarrow 0\}]\!] \vee [\![(f \wedge g)\{x \leftarrow 1\}]\!] \\
&= [\![f\{x \leftarrow 0\} \wedge g\{x \leftarrow 0\}]\!] \vee [\![f\{x \leftarrow 1\} \wedge g\{x \leftarrow 1\}]\!] \\
&= [\![f \wedge g\{x \leftarrow 0\}]\!] \vee [\![f \wedge g\{x \leftarrow 1\}]\!] \\
&= [\![f \wedge 0]\!] \vee [\![f \wedge 1]\!] \;=\; [\![f]\!]
\end{aligned}$$

□

Expansions as in Quantor [17] and long distance resolution as in Quaffle [12] allow some kind of reasoning across quantifier alternations. With expansions it is in principle possible to resolve clauses on outer variables that stem from different copies of the expanded clauses. Similarly, long distance resolution allows to compactly capture in a learned clause the effect of propagating information from an outer existential scope through an universal quantifier into an inner existential scope and back to the outer existential scope.

Currently we do not know how to efficiently simulate expansions and long distance resolution in our proof format. We conjecture that an even stronger proof system is necessary for this purpose.

[1] with respect to equivalence of propositional unquantified formulas [16].

Definition 3 (Model). *Let V_i be the set of variables in a formula that have a quantification level less than or equal to i and let E_i and A_i be the sets of existentially resp. universally quantified variables in V_i, i.e., $E_i \cup A_i = V_i$. A model M of the formula is then a set of functions*

$$M := \{f_{v_k} : \mathbb{B}^{k-1} \to \mathbb{B} \,|\, v_k \in E_n\},$$

where every f_{v_k} depends exactly on the $k-1$ variables from V_{k-1}.[2]

This definition is essentially the same as the one used by Kleine Büning [26]. It is also used in sKizzo [27] to certify satisfiable resp. valid instances. The functions f_v are also called *Skolem-functions*. Using our extension rule we can provide a model as a set of extensions to the formula by defining new variables representing the value of the Skolem functions. Our certificate thus contains a list of pairs of the form (v, f_v) where f_v is the fresh variable encoding the Skolem-function for v. With this approach, we have implemented model generation for two solvers, Ebddres and Squolem, presented in detail in Sect. 3. A general discussion for model generation with the extension rule for different QBF solvers is presented below.

As discussed above, new variables have to be ordered carefully. If they are simply quantified in the innermost scope, the corresponding function could depend on variables with higher quantification level than the one that it serves as a model for. For example, in a formula with the quantification sequence $\exists e \forall a \exists g$, the newly defined variable g may depend on the value of a, which may turn the defined function into an invalid model. Therefore our proof checker orders extension variables as low as possible, i.e., right after the variable with the highest quantification level occurring in the extension function. Checking that a model function does not depend on higher quantification levels is then trivial again.

However, checking the *validity* of a model according to Def. 3 is co-NP complete [26,28]. In general, our approach is to check each clause individually to be tautological, leveraging incremental SAT solver technology. This can be achieved by keeping the model functions in the SAT solver and adding the negation of a clause, i.e., the negations of the clauses literals, as assumptions. The resulting problem must then be unsatisfiable, which means that it is impossible to unsatisfy the given clause with the model provided. It is possible to provide a refutation proof for each of those sub-problems (or, alternatively, for the whole formula instead of separate clauses). The complexity of checking this (annotated) model for validity is then polynomial in the size of the certificate. So far, our certificates only contain the Skolem-functions. In future, we will study how providing a refutation for each clause in the model affects the model generation time and model size, as well as the model verification time.

In Skolemization-based solvers constructing a model is easy, since the solvers basic strategy is to construct a model; it just has to dump the Skolem-functions

[2] It is also possible to let the f_{v_k} only depend on the universally quantified variables of V_{k-1}. However, our definition may result in more compact representations of the functions f_{v_k}.

it generates. DPLL- and search-based solvers split on variables using different strategies. Assuming the formula is valid, the solver always encounters either unit clauses, and propagates this information, or splits on a new variable. Whenever a unit clause (l) occurs under some variable assignment α to other variables, we can read this as $\alpha \Rightarrow l$, which corresponds to one entry in the function table of the model function for the variable of l. This information can immediately be dumped as $x_i = a_1 \wedge ... \wedge a_n$, where x_i is a fresh variable and the a_i are the literals from α. When the evaluation is finished, the final model-function is $f_v = x_1 \vee \cdots \vee x_n$.

In Q-Resolution-based solvers, particularly [16,20], one can interpret any resolution between two clauses c_1 and c_2 into a resolvent as the generation of a conflicting clause. For instance, if c_1 contains a literal x and c_2 contains $\neg x$, then $\neg(c_1 \backslash \{x\}) \wedge \neg(c_2 \backslash \{\neg x\})$ may not happen, because x would have to be 1 and 0 at the same time to satisfy both clauses. One strategy to record a model is to start with overspecified functions and to refine them whenever resolution is applied. BDD-based solvers implementing the bucket algorithm [18] (and also EBDDRES) store intermediate BDDs for variables to be eliminated. From these, it is possible to dump Skolem-functions using the extension rule. The procedure for EBDDRES is presented in more detail in Sect. 3.

An alternative way of providing a model is to provide a refutation for the negation of a valid formula (which would then be invalid). Experiments that negate a formula by translation of the resulting DNF back to a CNF using the Tseitin-transformation have shown that this approach is infeasible. All of the problems that could be solved within 600 seconds by three different solvers (QUANTOR, SKIZZO, SQUOLEM) could either not be solved within the same time when inverted, or took considerably more time to solve.

3 Implementation

We implemented SQUOLEM, a *new* skolemization-based solver, which generates both models and refutations. SQUOLEM eliminates quantifiers from the inside out by explicitly generating Skolem-functions for existential variables. For each variable that is about to be eliminated, it collects all clauses in which the variable occurs and interprets them as implications, e.g., $(a \vee b)$ is interpreted as $\neg a \rightarrow b$. The set of these implications essentially forms a function, by which the variable is replaced.

In case of conflicting implications, e.g., $\neg a \rightarrow b$ and $\neg a \rightarrow \neg b$, a clause stating that this case cannot happen, e.g., (a), is added. In terms of the resolution calculus, the conflict clause can directly be obtained as a resolvent from the two conflicting implications. The process is iterated until either a complete model has been constructed, or conflicting unit clauses occur. In the first case we output the model, in the second case we output a q-resolution trace constructed from the information about a clause's parents, which we record during model construction.

We have instrumented two other *already existing* solvers, (i) the BDD-based solver EBDDRES [29,30] and (ii) the search-based solver QUAFFLE [12]. EBDDRES

produces both models and refutation traces whereas with QUAFFLE we are so far limited to refutation traces. Unfortunately, in case of QUAFFLE we also had to disable learning, since we cannot trace long distance resolution steps, as already discussed above. EBDDRES is a BDD-based QBF solver that eliminates variables starting from the innermost scope. In order to eliminate a variable x, EBDDRES first builds a conjunction of all the clauses containing x and then quantifies x using one standard BDD OR- resp. AND-operation if the variable is existential resp. universal. After all variables are eliminated, a constant BDD is obtained. For simplicity, the variable ordering for the BDDs is the reverse of the QBF variable order. Thus, root variables are always eliminated.

EBDDRES produces refutations by introducing a Tseitin variable for each BDD node. This Tseitin variable is defined to be an if-then-else gate. Given a BDD node n, let x be its variable and t, t_0, and t_1 the Tseitin variables introduced for n, its left child, and its right child, respectively. Then the definition of t is $t \Leftrightarrow (x \ ? \ t_1 \ : \ t_0)$. Thus, if x is true (false), the Tseitin variable t_1 (t_0) has to be true. The refutation is constructed by first showing that the Tseitin variables for the root nodes of the BDDs for every original clause have to be true. Then the logical operations of the solving algorithm are traced until it is shown that the Tseitin variable for the constant BDD zero has to be true, a contradiction. All the details except for universal quantification can be found in [29, 30]. For universal quantification, the proof rule is as follows. Let x be a universal variable to be eliminated and t the variable corresponding to the root node of the BDD which is the conjunction of all the clauses containing x. We have derived that t must be true. The definition of t introduces (among others) the clause $(\neg t \vee x \vee t_0)$. Resolving this with (t) yields $(x \vee t_0)$ which, based on Def. 1, can be forall-reduced to (t_0) since t_0 is not in the scope of x. The proof for t_1 is similar.

EBDDRES produces models as follows. In the bucket algorithm, when variable x is to be eliminated, a BDD containing precisely all the constraints on x is built. Going from the root to the child where x is true (right child) gives another BDD that encodes all the valuations where x has to be true to satisfy all the constraints. If x is existential, then this is precisely a Skolem-function for x. Thus the certificate consists of definitions of the Tseitin variables for this BDD and the Skolem-function is set to be equivalent to Tseitin variable of the root (analogously, we could have chosen the negation of the left child).

For the search-based solver QUAFFLE, the refutation is constructed as follows. Assume (without loss of generality) that the innermost scope is existential. If the instance is unsatisfiable, both choices for the truth value of an existential variable lead to a conflict. For a universal variable, at least one choice leads to a conflict. Whenever a conflict is reached, a conflict clause can be derived.

We produce refutation proofs from these conflict clauses. Consider for instance the simple search tree example presented in Fig. 1 and let the solid lines denote paths to conflicts. Let the left (right) arrow from a node mean setting the truth value of a variable to false (true). Now, the leftmost path $(\neg x \wedge \neg y \wedge \neg z)$ leads to a conflict producing the conflict clause $(x \vee y \vee z)$. Similarly, the path $(\neg x \wedge \neg y \wedge z)$

produces the clause $(x \vee y \vee \neg z)$. We resolve these, and forall-reduce the result $(x \vee y)$ and thus obtain the clause (x). Similarly, from the conflict clauses from the right-hand side (where x is true) we get $(\neg x)$ and resolving the two, the empty clause. The above representation is simplified. Our experimental work has revealed that in most cases not a complete conflict clause is obtained but one that subsumes it. The consequence of this is that it is possible to omit some resolution steps.

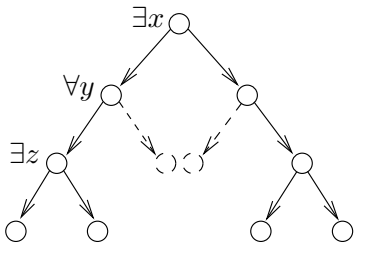


Fig. 1. QUAFFLE search tree

If a formula has a large alternation depth, our proof-generation algorithm starts by resolving the literals from the innermost existential scope. Subsequently, the new clauses are forall-reduced to eliminate the enclosing universal scope. Then, the proof generation eliminates the second innermost existential scope and thus alternates between resolution and forall-reduction until the outermost scope.

Note that we have only been able to instrument QUAFFLE without learning. So far, it is an open problem how to handle long-distance resolution in the presented framework. Secondly, we are not able to create certificates, only refutation traces.

To verify the certificates that we extract, we implemented QBV, the Quantified Boolean Verifier. The implemented algorithm executes applications of our extension rules and q-resolutions, as listed in the certificate. The last statement in a certificate is a conclusion line that either provides the index of an empty clause (for refutations), or a list of equivalences of variables for a model. As suggested in [27], we check every clause separately against the model. For this purpose we use MINISAT (Version 1.14p) in incremental mode, i.e., we load the model into MINISAT and then add the negation of a clause as an assumption, which must result in an unsatisfiable problem. As stated earlier, this problem is in general Co-NP complete. In a future version we will provide support for refutations for every clause in the original formula, such that the complexity of this step becomes polynomial (with adverse effects on the certificate generation time).

4 Experimental Results

To show that certificate extraction is feasible we have conducted experiments on the 2005 fixed instance and the 2006 preliminary QBF-Eval[3] datasets, in total 445 test cases. We used EBDDRES, QUAFFLE, and SQUOLEM to generate certificates, and verified them using QBV.

The tests for EBDDRES and QUAFFLE were run on a cluster of Intel Pentium IV PCs (3 GHz) with 2 GB RAM each. We set a time limit of 600 seconds and a memory limit of 1 GB. EBDDRES is able to create a model for 80 instances and a refutation trace for 86 instances. For QUAFFLE (without learning), 26

[3] `http://www.qbflib.org/`

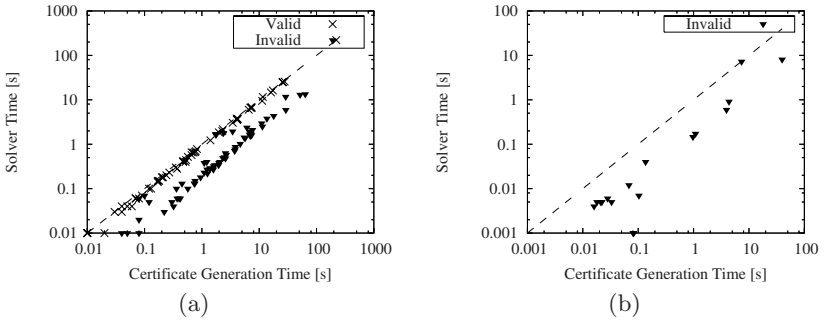


Fig. 2. Certificate Generation Time vs. Solver Time for (a) EBDDRES and (b) QUAFFLE

instances could be refuted. We first compare the time required to solve an instance to the time needed to create the certificate. The results for EBDDRES and QUAFFLE are shown in Figs. 2(a) and 2(b), respectively. They show that for both solvers, generating a certificate takes longer than solving the instance. For EBDDRES the overhead for models is about 15% and for refutations 440%. This behavior is probably due to the fact that the refutation files for EBDDRES are so large, that merely saving them takes a lot of time. The same overhead is also observed in the propositional case [29, 30].

For QUAFFLE the overhead is even higher, 1440%. However, this result is not illustrative since we were only able to solve very few instances and these instances belong to only 6 families. For one instance, 'k_lin_p-4', the trace generation overhead is only 1.6%.

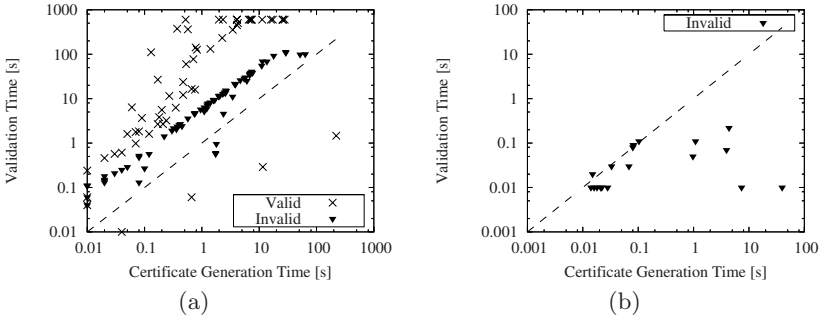


Fig. 3. Certificate Generation Time vs. Validation Time (a) EBDDRES and (b) QUAFFLE

We also compare the time needed to validate certificates to the time it takes to generate them. The results for EBDDRES and QUAFFLE are shown in Figs. 3(a) and 3(b). Our experimental results show that, for EBDDRES, it takes on average longer to validate a certificate; again the overhead depends heavily on whether the instance is valid or invalid. Validating models, takes on average over 100 times longer than to generate them. QBV actually times out on 15 instances

(time limit 600 seconds) where model generation is feasible. Refutations, on the other hand, can be verified quicker, the ratio is 5.4. For QUAFFLE, the trace validation times are neglible (on average < 0.04 seconds), as Fig. 3(b) shows. The observed behavior is in line with the fact that verifying a model is in general Co-NP complete whereas a resolution trace can be verified polynomially.

The tests for SQUOLEM were run on an Intel Xeon 3.0 GHz machine with 4 GB RAM. Out of 445 instances in the original dataset, our solver finishes on 142 within 600 seconds and a memory limit of 1 GB; 73 instances were found to be valid, 69 invalid.

First we compare the time to solve an instance to the time needed to generate a certificate. Naturally, in a purely Skolemization-based solver, there is a small difference in those times. Fig. 4(a) shows that the overhead of certificate generation is usually very small (on average 3.5% for models and 4.5% for refutations, with respect to solving time).

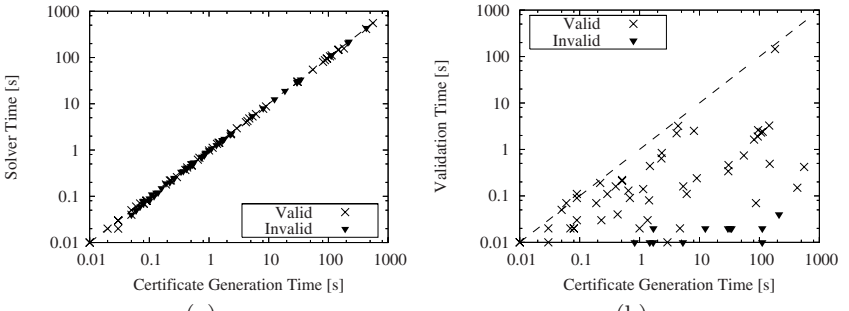


Fig. 4. (a) Certificate Generation Time vs. Solver Time and (b) Certificate Generation Time vs. Validation Time for SQUOLEM

As a second experiment, we investigate the time taken to validate a certificate. Fig. 4(b) shows that the time taken to validate a refutation is negligibly small (on average < 0.01 seconds); certificates of validity, on the other hand, take on average 2.38 seconds to validate. The main contribution to this time is the instance 'qshifter_7', which takes 144 seconds to validate. Excluding this instance, the average validation time is 0.4 seconds. The reason for this extraordinary high runtime on just this single instance is that MINISAT actually runs into a hard problem: 97% of the runtime is spent on the final model validation.

An interesting property of certificates is the size of the generated files. Table 1 gives an overview of the relative size of the generated certificates, with respect to the size of the original formula file. We present the traces in the original ASCII format as well as compressed with `gzip`. The data indicates that the certificates generated by EBDDRES are very large compared to QUAFFLE and SQUOLEM. Furthermore, its refutations (tracing complex BDD operations) are larger than its models. For SQUOLEM, on the other hand, the numbers suggest that models

Table 1. Relative size of the generated certificates

		Normal			Compressed		
Solver	Type	Best	Avg.	Worst	Best	Avg.	Worst
EBDDRES	Valid	<0.1	210.9	3304.3	<0.1	52.3	784.8
	Invalid	1.0	6857.6	145414.0	1.0	1594.3	31948.3
QUAFFLE	Valid	-	-	-	-	-	-
	Invalid	<0.1	3.4	17.0	<0.1	1.3	5.0
SQUOLEM	Valid	0.7	10.1	175.6	0.2	2.8	49.6
	Invalid	<0.1	3.3	55.0	<0.1	0.8	10.4

are considerably larger than refutations. The certificate format is not optimized for file size, but as the data indicates, the files compress well with `gzip`, if smaller files are required.

Finally, we compare our solvers with two state-of-the-art solvers, QUANTOR and sKIZZO. As QUAFFLE only produces refutation traces, we have no data on valid instances for this solver. Thus, we chose to provide the number of invalid instances that each solver is able to solve within 600 seconds and a 1 GB memory limit, from the total 205 in the data set. The numbers in Tbl. 2 indicate that our solvers together can solve about half as many instances.

Table 2. The number of solved instances in the test set

	EBDDRES	QUAFFLE	SQUOLEM	QUANTOR	sKIZZO
Solved Instances	80	26	69	153	175
Solved Inst. (Union)	90			178	

5 Reference Implementations

EBDDRES, the instrumented version of QUAFFLE, and the exprimental data can be downloaded from `http://fmv.jku.at/ebddres`. SQUOLEM, the certificate validator QBV, the experimental data, and a formal specification of the supported certificate format are available at `http://www.verify.ethz.ch/qbv`.

6 Conclusion

We show that it is possible to define a proof format for QBF that is applicable to a wide range of different QBF solvers. Nevertheless, important common features of other QBF solvers cannot be traced efficiently in this format. Even though it seems that existential expansion steps of structural QBF solvers can be traced with our format, we do not know how to trace universal expansion steps with the current set of rules. The same applies to long distance resolution. This is in contrast to SAT, where just adding the extension rule generates a proof calculus, which is as powerful as any other known propositional proof system.

References

1. Stockmeyer, L.J.: The polynomial–time hierarchy. TCS **3** (1976) 1–22
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
3. Rintanen, J.: Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research **10** (1999) 323–352
4. Otwell, C., Remshagen, A., Truemper, K.: An effective QBF solver for planning problems. In: MSV/AMCS, CSREA Press (2004) 311–316
5. Giunchiglia, E., Narizzano, M., Tacchella, A.: QBF reasoning on real–world instances. In: Proc. of SAT. LNCS 3542, Springer (2004) 105–121
6. Ladner, R.E.: The computational complexity of provability in systems of modal propositional logic. SIAM Journal on Computing **6**(3) (1977) 467–480
7. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic model checking without BDDs. In: Proc. of TACAS. LNCS 1579, Springer (1999) 193–207
8. Dershowitz, N., Hanna, Z., Katz, J.: Bounded model checking with QBF. In: Proc. of SAT. LNCS 3569, Springer (2005) 408–414
9. Jussila, T., Biere, A.: Compressing BMC encodings with QBF. In: Proc. 4th Intl. Work. on Bounded Model Checking (BMC). To be published in ENTCS, Elsevier (2006)
10. Benedetti, M.: Experimenting with QBF-based formal verification. In: Proc. of the 3rd International Workshop on Constraints in Formal Verification (CFV). To be published in ENTCS, Elsevier (2005)
11. Plaisted, D.A., Biere, A., Zhu, Y.: A satisfiability procedure for quantified boolean formulae. Discrete Appl. Math. **130**(2) (2003) 291–328
12. Zhang, L., Malik, S.: Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In: Proc. of CP. LNCS 2470, Springer (2002) 200–215
13. Letz, R.: Lemma and model caching in decision procedures for quantified boolean formulas. In: Proc. of TABLEAUX. LNCS 2381 (2002) 160–175
14. Giunchiglia, E., Narizzano, M., Tacchella, A.: QUBE: A system for deciding quantified boolean formulas satisfiability. In: Proc. of IJCAR. LNCS 2083, Springer (2001) 364–369
15. Cadoli, M., Giovanardi, A., Schaerf, M.: An algorithm to evaluate quantified boolean formulae. In: Proc. of AAAI/IAAI, AAAI (1998) 262–267
16. Samulowitz, H., Bacchus, F.: Using SAT in QBF. In: Proc. of CP. LNCS 3709, Springer (2005) 578–592
17. Biere, A.: Resolve and expand. In: Proc. of SAT. LNCS 3542, Springer (2004) 59–70
18. Pan, G., Vardi, M.Y.: Symbolic decision procedures for QBF. In: Proc. of CP. LNCS 3258, Springer (2004) 453–467
19. Benedetti, M.: Evaluating QBFs via symbolic skolemization. In: Proc. of LPAR. LNCS 3452. Springer (2005) 285–300
20. Samulowitz, H., Bacchus, F.: Binary clause reasoning in QBF. In: Proc. of SAT. LNCS 4121, Springer (2006)
21. Narizzano, M., Tacchella, A., Pulina, L.: Report of the third QBF solvers evaluation. JSAT **2** (2006) 145–164
22. Tseitin, G.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic **2** (1968) 115–125

23. Benedetti, M.: sKizzo: A suite to evaluate and certify QBFs. In: Proc. of CADE. LNCS 3632, Springer (2005) 369–376
24. Yu, Y., Malik, S.: Validating the result of a quantified boolean formula (QBF) solver: theory and practice. In: Proc. of ASP-DAC, ACM Press (2005) 1047–1051
25. Kleine Büning, H., Karpinski, M., Flügel, A.: Resolution for quantified boolean formulas. Inf. Comput. **117**(1) (1995) 12–18
26. Kleine Büning, H., Zhao, X.: On models for quantified boolean formulas. In: Logic versus Approximation. LNCS 3075, Springer (2004) 18–32
27. Benedetti, M.: Extracting certificates from quantified boolean formulas. In: Proc. of IJCAI. (2005) 47–53
28. Büning, H.K., Subramani, K., Zhao, X.: On boolean models for quantified boolean horn formulas. In: Proc. of SAT. LNCS 2919, Springer (2003) 93–104
29. Sinz, C., Biere, A.: Extended resolution proofs for conjoining BDDs. In: Proc. of the 1st Intl. Computer Science Symp. in Russia (CSR 2006). LNCS 3967, Springer (2006) 600–611
30. Jussila, T., Sinz, C., Biere, A.: Extended resolution proofs for symbolic SAT solving with quantification. In: Proc. of SAT. LNCS 4121, Springer (2006) 54–60

Dynamically Partitioning for Solving QBF

Horst Samulowitz and Fahiem Bacchus

Department of Computer Science, University of Toronto, Canada
{horst,fbacchus}@cs.toronto.edu

Abstract. In this paper we present a new technique to solve Quantified Boolean Formulas (QBF). Our technique applies the idea of dynamic partitioning to QBF solvers. Dynamic partitioning has previously been utilized in #SAT solvers that count the number of models of a propositional formula. One of the main differences with the #SAT case comes from the solution learning techniques employed in search based QBF solvers. Extending solution learning to a partitioning solver involves some considerable complexities which we show how to resolve. We have implemented our ideas in a new QBF solver, and demonstrate that dynamic partitioning is able to increase the performance of search based solvers, sometimes significantly. Empirically our new solver offers performance that is superior to other search based solvers and in many cases superior to non-search based solvers.

1 Introduction

The variables of a SAT problem are implicitly existentially quantified: SAT asks the question "does there exist a setting of these variables that satisfies the formula?" QBF is a generalization of SAT in which the variables are allowed to be universally as well as existential quantified: QBF asks the question "is the formula true for all settings of the universal variables." The ability to nest universal and existential quantification in arbitrary ways makes QBF considerable more expressive than SAT. While any NP problem can be encoded in SAT, QBF allows us to encode any PSPACE problem: QBF is PSPACE-complete.

This expressiveness opens a much wider range of potential application areas for a QBF solver, including areas like automated planning, non-monotonic reasoning, electronic design automation, scheduling, and model checking and verification, e.g., [1,2,3]. The difficulty, however, is that QBF is in practice a much harder problem to solve than SAT. (It is also much harder theoretically assuming that PSPACE $\neq$ NP). One indication of this practical difficulty is the fact that current QBF solvers are typically limited to problems that are about 1-2 orders of magnitude smaller than the instances solvable by current SAT solvers (1000's of variables rather than 100,000's).

Nevertheless, this limitation in the size of the instances solvable by current QBF solvers is somewhat misleading. In particular, many problems have a much more compact encoding in QBF than in SAT. For example, in [4] the authors give an innovative application of QBF to hardware debugging, showing that the QBF encoding of the problem is many times smaller than an equivalent SAT encoding. Results like this demonstrate the potential of QBF and the importance of further improving QBF solvers.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 215–229, 2007.

In this paper we present a new technique for improving QBF solvers. Our technique extends the idea of dynamic partitioning, prominently utilized in #SAT solvers, to make it useful in a QBF solver. #SAT is the problem of counting the number of models of a CNF formula, and the idea of dynamic partitioning for solving #SAT was first utilized in [5]. That work presented a DPLL based algorithm for #SAT which examined the remaining CNF theory at each node of the search tree. The algorithm tried to partition the remaining theory into disjoint components that shared no variables. The disjoint components could then be solved independently of each other, resulting in a significant improvement in run time. In particular, since the run time is worst case exponential in the number of variables, partitioning can move us from $O(2^n)$ to $kO(2^{n/k})$ if the problem can be broken into k equally sized partitions. Applying this recursively can potentially yield an exponential speed up. See [6,7] for more detailed theoretical results characterizing the speedups that can be achieved from partitioning.

Here we apply dynamic partitioning to QBF. We first make the observation that a QBF theory can be partitioned into independent components as long as these components share no *existential* variables. That is, QBF components do not have to be completely disjoint as is the case with #SAT, just so long as they are existentially disjoint. We then show how clause learning in search based QBF solvers can be quite easily extended to deal with partitioning. Extending cube (solution) learning to deal with partitioning is considerable more complex, and is perhaps the key innovation of our work. We have implemented our ideas in a new QBF solver 2clsP. 2clsP is built on top of the 2clsQ [8] solver, which with the addition of some preprocessing techniques was the top scoring solver in the 2006 QBF competition. We show empirically that these new ideas yield a significant improvement in 2clsQ's performance. We also demonstrate that 2clsP offers performance that is superior to other search based solvers and in many cases superior to non-search based solvers like Quantor [9] and Skizzo [10]. These results underscore the potential that partitioning, when properly augmented with clause and cube learning, has for helping us improve current QBF solvers.

In the sequel we first present some necessary background, setting the context for our methods. We then present the details of how clause and particularly cube learning can be extended to partitioning. Then we provide empirical evidence of the effectiveness of our approach, and close with a discussion of future work and some conclusions.

2 Background

A quantified boolean formula has the form $\boldsymbol{Q}.F$, where F is a propositional formula expressed in CNF and $\boldsymbol{Q}$ is a sequence of quantified variables ($\forall x$ or $\exists x$). We require that no variable appear twice in $\boldsymbol{Q}$ and that all variables in F appear in $\boldsymbol{Q}$ (i.e., F contains no free variables). $\boldsymbol{Q}$ may have extra variables not mentioned in F. Such variables can be removed or retained—they do not affect the truth of the QBF.

QBF solvers are interested in answering the question of whether or not $\boldsymbol{Q}.F$ expresses a true or false assertion, i.e., whether or not $\boldsymbol{Q}.F$ is true or false. The **reduction** of a CNF formula F by a literal ℓ is the new CNF $F|_\ell$ which is F with all clauses containing ℓ removed and $\neg\ell$, the negation of ℓ, removed from all remaining clauses.

The reduction of F by a **set** of literals L is defined to be the sequential reduction of F by each literal in L. It can easily be observed that the final reduced CNF is independent of the order these reductions are performed.

1. If F is the empty set of clauses then $\boldsymbol{Q}.F$ is *true*.
2. If F contains an empty clause then $\boldsymbol{Q}.F$ is *false*.
3. $\forall v\boldsymbol{Q}.F$ is *true* iff both $\boldsymbol{Q}.F|_v$ and $\boldsymbol{Q}.F|_{\neg v}$ are true.
4. $\exists v\boldsymbol{Q}.F$ is *true* iff at least one of $\boldsymbol{Q}.F|_v$ and $\boldsymbol{Q}.F|_{\neg v}$ is true.

A **quantifier block** qb of $\boldsymbol{Q}$ is a maximal contiguous subsequence of $\boldsymbol{Q}$ where every variable in qb has the same quantifier type. We order the quantifier blocks by their sequence of appearance in $\boldsymbol{Q}$: $qb_1 \leq qb_2$ iff qb_1 is equal to or appears before qb_2 in $\boldsymbol{Q}$. Each variable x in F appears in some quantifier block denoted by $qb(x)$.

Definition 1

1. *For two variables x and y, $x \leq_q y$ if $qb(x) \leq qb(y)$ and $x <_q y$ if $qb(x) < qb(y)$.*
2. *Variable x is **universal** (**existential**) if its quantifier in $\boldsymbol{Q}$ is $\forall$ ($\exists$).*
3. *A variable x is **downstream** (**upstream**) of a set of variables V if (1) $x \notin V$ and (2) $\forall y.y \in V \rightarrow y \leq_q x$ ($\forall y.y \in V \rightarrow x \leq_q y$)). That is, x is not a member of V and appears no sooner (later) in the quantifier sequence $\boldsymbol{Q}$ than the last (first) quantifier block containing elements of V.*
4. *A variable x is **maximal** (**minimal**) in a set of variables V if (1) $x \in V$ and (2) $\forall y.y \in V \rightarrow y \leq_q x$ ($\forall y.y \in V \rightarrow x \leq_q y$). That is x is a member of V and appears in the highest (lowest) quantifier block amongst all variables of V.*

As a slight abuse of notation we often use a literal ℓ to refer to ℓ's variable. For example, when we say that ℓ is maximal in a set of variables V, we mean that ℓ's variable is maximal in V. Similarly, we might assert that ℓ is universal if ℓ's variable is universal, that $\ell_1 <_q \ell_2$ if ℓ_1's variable is $<_q$ than ℓ_2's variable, or that ℓ is added to a set of variables V if ℓ's variable is added to V.

For example, $\exists e_1e_2.\forall u_1u_2.\exists e_3e_4.(e_1, \neg e_2, u_2, e_4) \wedge (\neg u_1, \neg e_3)$ is a QBF with $\boldsymbol{Q} = \exists e_1e_2.\forall u_1u_2.\exists e_3e_4$ and F equal to the two clauses $(e_1, \neg e_2, u_2, e_4)$ and $(\neg u_1, \neg e_3)$. The quantifier blocks in order are $\exists e_1e_2$, $\forall u_1u_2$, and $\exists e_3e_4$, and we have, e.g., that, $e_1 <_q e_3$, $u_1 <_q e_4$, u_1 is universal, e_4 is existential, e_4 is downstream of the set $\{u_2, e_3\}$, e_3 is maximal in the set $\{u_2, e_3\}$, and u_2 is upstream of the set $\{u_1, e_3, e_4\}$.

We make two useful observations about QBFs (an easy proof is given, e.g., in [11]).

Observation 1

A. *If $F \vDash F'$ then $\boldsymbol{Q}.F \Rightarrow \boldsymbol{Q}.F'$. That is, if every SAT model of F is also a SAT model of F' then if $\boldsymbol{Q}.F$ is true $\boldsymbol{Q}.F'$ must also be true. Note that this holds even when F' contains a superset of F's variables.*

B. *A universal variable u is called a **tailing universal** in a clause c if for every existential variable $e \in c$ we have that $e <_q u$. The universal reduction [12] of a clause c is the process of removing all tailing universals from c. If F' is the result of applying universal reduction to some clause of F, then $\boldsymbol{Q}.F \Leftrightarrow \boldsymbol{Q}.F'$.*

2.1 Partitioning QBF

Now we discuss the conditions under which a QBF can be partitioned into a conjunction of independent sub-formulas. First we recall two standard logical laws for quantifiers. Let Φ_1 and Φ_2 be propositional formulas.

1. If Φ_1 does not contain x then $\exists x.(\Phi_1 \wedge \Phi_2) \Leftrightarrow (\Phi_1 \wedge \exists x.\Phi_2)$ and $\forall x.(\Phi_1 \wedge \Phi_2) \Leftrightarrow (\Phi_1 \wedge \forall x.\Phi_2)$.
2. $\forall x.(\Phi_1 \wedge \Phi_2) \Leftrightarrow (\forall x.\Phi_1 \wedge \forall x.\Phi_2)$

Observation 2. *If F is a CNF formula that can be divided into two CNF's F_1 and F_2 such that the clauses in F_1 and F_2 share no existential variables, then $\boldsymbol{Q}.F \Leftrightarrow \boldsymbol{Q}_1.F_1 \wedge \boldsymbol{Q}_2.F_2$, where $\boldsymbol{Q}_i$ is the subsequence of $\boldsymbol{Q}$ containing only the variables of F_i.*

To see that this is true we first rewrite $\boldsymbol{Q}.F$ as $\boldsymbol{Q}.(F_1 \wedge F_2)$, then we proceed to use the above logical laws to distribute the variables of $\boldsymbol{Q}$ to F_1 or F_2, starting with the innermost quantified variables of $\boldsymbol{Q}$. We can apply this observation multiple times to separate $\boldsymbol{Q}.F$ into a conjunction of k smaller QBFs.

For example,

$$\begin{aligned}
&\forall u_1 \exists e_1 \forall u_2 \exists e_2 e_3.\big((u_1, \neg e_1) \wedge (u_2, \neg e_2) \wedge (u_2, e_3)\big)\\
&\Leftrightarrow \forall u_1 \exists e_1 \forall u_2 \exists e_2.\big((u_1, \neg e_1) \wedge (u_2, \neg e_2) \wedge \exists e_3.(u_2, e_3)\big)\\
&\Leftrightarrow \forall u_1 \exists e_1 \forall u_2.\big((u_1, \neg e_1) \wedge \exists e_2.(u_2, \neg e_2) \wedge \exists e_3.(u_2, e_3)\big)\\
&\Leftrightarrow \forall u_1 \exists e_1.\big((u_1, \neg e_1) \wedge \forall u_2 \exists e_2.(u_2, \neg e_2) \wedge \forall u_2 \exists e_3.(u_2, e_3)\big)\\
&\Leftrightarrow \forall u_1.\big(\exists e_1.(u_1, \neg e_1) \wedge \forall u_2 \exists e_2.(u_2, \neg e_2) \wedge \forall u_2 \exists e_3.(u_2, e_3)\big)\\
&\Leftrightarrow \forall u_1 \exists e_1.(u_1, \neg e_1) \wedge \forall u_2 \exists e_2.(u_2, \neg e_2) \wedge \forall u_2 \exists e_3.(u_2, e_3)
\end{aligned}$$

2.2 Partitioning for a Search Based QBF Solver

Observation 2 immediately yields a partitioning search based QBF solver.

```
1  QBF-Prt(Q.F)
2  if F contains an [empty clause/is empty] then
3      return([FALSE/TRUE])
4  for ℓ ∈ {v, v̄} for some v ∈ F with outermost scope in Q do
5      success = TRUE
7      Partitions = Partition(Q.F|ℓ)
8      foreach Q_i.P_i ∈ Partitions while success do
10         if QBF-Prt(Q_i.P_i) = FALSE then
11             success = FALSE
12     if [¬success/success] AND v is [universal/existential] then
13         return[FALSE/TRUE]
14 if v is [universal/existential] then
15     return[TRUE/FALSE]
```

That is, we branch on variables respecting the order of quantification, just as in a standard search based QBF solver. However, after every variable has been instantiated (at this stage some propagation can also be preformed to further reduce the remaining theory) we check if the remaining theory can be broken up into existentially disjoint partitions (line 7). This can be accomplished in time linear in the size of the remaining theory with a simple depth-first search or a union-find algorithm. We then solve these partitions independently (line 10). Since the remaining theory is equivalent to the conjunction of these partitions, they must all be true for the entire theory to be true. Hence, we can stop if any of these partitions is false.

Unfortunately, although partitioning is a good idea, our empirical investigations allowed us to conclude that this simple version of partitioning is completely ineffective in practice. Fundamental to the performance of search based QBF solvers are the techniques of clause and cube learning [13,14]. Without these techniques a partitioning solver performs much worse than current search based solvers. One of the key contributions of our work is to show how learning can be extended so that it can be applied in the context of partitioning.

2.3 Quantifier Trees

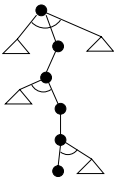
In [15] Benedetti uses the logical laws for quantifiers mentioned above to build a Quantifier Tree for a QBF. The quantifier tree specifies, among other things, a *static* decomposition of the QBF. That is, it specifies a decomposition that ignores the truth value assigned to each variable. Benedetti also points out that such trees can be used in a partitioning search based QBF solver similar to **QBF-Prt** presented above. There are two main differences between this work and what we present here. First, as noted above the simple notion of of partitioning presented in **QBF-Prt** is ineffective without learning. As we will see adding learning to partitioning is a non-trivial new contribution of our work. Second, the partitioning algorithm presented in **QBF-Prt** employs *dynamic* partitioning. That is, the partitioning generated when we set $v =$ TRUE can be entirely different to the partitioning generated when $v =$ FALSE. In a quantifier tree the partitioning will be the same for both truth values. Since this difference compounds as we set more variables this means that the partitions generated dynamically can be considerably more refined than those specified in a static quantifier tree.[1]

3 Learning with Partitioning

Search based QBF solvers employ the powerful techniques of clause and cube (solution) learning [13,14]. These techniques are essential for obtaining good performance from a search based QBF solver. In this section we show how learning can be used with partitioning.

[1] In [7] it was shown that for #SAT dynamic partitioning can yield a super-polynomial speedup over *any* static decomposition on some instances. We suspect that a similar result holds for QBF, but this is not yet proven.

To facilitate the subsequent discussion the figure on the right shows a sample path in the **QBF-Prt** search tree. The black circles correspond to literals made true along the current path, arcs connecting branches indicate points where the theory was split into partitions, and the triangles correspond to the other partitions that were generated along this path. The partitions on the left of the current path have already been solved, while those on the right of the current path remain to be solved. We call the partitions that lie off of the current path *inactive*, and the partition currently being solved *active*.



3.1 Clause Learning

For the most part clause learning can be used without modification in a partitioning solver. For example, if the current path leads to a conflict a conflict clause can be learned and universal reduction applied—the conflict must be a subset of the literals set along the current path. This conflict can then be used to backtrack as least far enough to undo the conflict, as below this point no solution exists for the active partition. Since the theory is the conjunction of its partitions, the status of the inactive partitions we backtrack past is irrelevant—falsifying the active partition is sufficient to falsify the entire theory. Note that backtracking further is also possible, e.g., backtracking to the 1st-UIP point. The search will continue as before from that backtrack point. Similarly, the learnt clauses can then be used in unit propagation as they normally would be in a non-partitioning solver.

The main subtleties in using clause learning with partitioning have already been addressed in [16] who showed how to use clause learning and partitioning in the context of solving #SAT. It is not difficult to show that their insights also hold for QBF. In particular, first, we are allowed to ignore the learned clauses when partitioning the theory since the learned clauses are entailed by the original theory. Second, it is sound to ignore existentials from inactive partitions that might be forced by the learned clauses. Alternatively we can allow them to be forced: any conflict generated by them will still be a valid conflict.

3.2 Cube Learning

In order to extend cube learning to allow partitioning we must first develop a new formalization of cube learning.

We first define the **restriction** of a clause c to a set of variables V to be the new clause c' formed by restricting c to the variables in V, i.e., removing from c all variables not mentioned in V. For example, $\textbf{\textit{restrict}}((x, \neg y, \neg z), \{x, y, w, t\}) = (x, \neg y)$, where the literal $\neg z$ has been removed since its variable z is not in the set $\{x, y, w, t\}$. We restrict a CNF formula F, $\textbf{\textit{restrict}}(F, V)$, by restricting each of its clauses. Note that if V contains all of the variables in c then $\textbf{\textit{restrict}}(c, V) = c$, and similarly $\textbf{\textit{restrict}}(F, V) = F$ if V contains all variables in F. We say that a QBF $\boldsymbol{Q}.F$ is **satisfied by the variables** V if the QBF $\boldsymbol{Q}.\textbf{\textit{restrict}}(F, V)$ is true.

We observe some facts about restriction and its relationship with reduction (setting a literal to be true).

Observation 3

1. *If* $V \subseteq V'$, *then* ***restrict***$(F, V) \vDash$ ***restrict***(F, V').
2. *If* $\boldsymbol{Q}.F$ *is satisfiable by any set of variables* V, *then it must also be true.*
3. *If* $\ell \notin V$ *then* ***restrict***$(F, V \cup \{\ell\})|_\ell$ *is equal to* ***restrict***$(F|_\ell, V)$.
4. *If* $\ell \notin V$ *then* ***restrict***$(F, V) \vDash$ ***restrict***$(F|_\ell, V)$.

Proof: For item 1, every clause of ***restrict***(F, V') is a superclause of a clause in ***restrict***(F, V). For item 2, this follows from item 1 and Observation 1.A by taking V' to be any superset of V that contains all variables of F. For item 3, this can be shown by considering what happens to every clause c of F under the stated sequence of reductions and restrictions. There are three cases to consider (a) $\ell \in c$, (b) $\neg\ell \in c$ and (c) all other clauses. For item 4, using the same three cases it can be shown that ***restrict***$(F|_\ell, V)$ contains a subset of the clauses of ***restrict***(F, V). ■

Definition 2. *A* ***cube*** *for the formula* $\boldsymbol{Q}.F$ *is a set of literals* ρ *and a set of variables* V *such that (a)* $\boldsymbol{Q}.F|_\rho$ *is satisfied by the variables* V, *and (b) the variables of* V *are all downstream of the variables of* ρ. *We write* $\mathbf{cube}[\rho, V, F]$ *to indicate that* ρ *and* V *is a cube for* $\boldsymbol{Q}.F$.[2]

In other words $\mathbf{cube}[\rho, V, F]$ iff $\boldsymbol{Q}$***.restrict***$(F|_\rho, V)$ is true, and V is downstream of ρ. This definition differs from the standard definition of a cube mainly in its introduction of the set of downstream variables V.

The following theorem justifies standard cube learning in a non-partitioning QBF solver.

Theorem 1

1. *If* π *is a set of literals that satisfies every clause of* F, *then* $\mathbf{cube}[\pi, \{\}, F]$.
2. *If* $\mathbf{cube}[\rho, V, F]$ *and* ℓ *is existential and maximal in* ρ, *then* $\mathbf{cube}[\rho - \{\ell\}, V \cup \{\ell\}, F]$
3. *If* $\mathbf{cube}[\rho_1, V_1, F]$ *and* $\mathbf{cube}[\rho_2, V_2, F]$ *are cubes such that (1) there is a unique literal* ℓ *such that* $\{\ell, \neg\ell\} \subseteq \rho_1 \cup \rho_2$, *(2) this clashing literal is universal, (3)* ℓ *is maximal in* $\rho_1 \cup \rho_2$, *and (4)* $V_1 \cup V_2$ *is downstream of* $\rho_1 \cup \rho_1$, *then* $\mathbf{cube}[\rho_1 \cup \rho_2 - \{\ell, \neg\ell\}, V_1 \cup V_2 \cup \{\ell\}, F]$.

Proof: For item 1, we see that $\boldsymbol{Q}.F|_\pi$ is an empty set of clause, thus it is satisfiable by any set of variables. **For item 2**, we know that $\boldsymbol{Q}$***.restrict***$(F|_\rho, V)$ is true, the claim is that $\Gamma =$ $\boldsymbol{Q}$***.restrict***$(F|_{\rho-\{\ell\}}, V \cup \{\ell\})$ is true and that $V \cup \{\ell\}$ is downstream of $\rho - \{\ell\}$. Since $\ell \in \rho$ it must be upstream of all of the variables in V by the definition of a cube. Hence, ℓ appears in the outermost quantifier block among the variables in Γ and by definition Γ is true iff $\Gamma|_\ell$ or $\Gamma|_{\neg\ell}$ are true. In fact, $\Gamma|_\ell =$ is true: $\Gamma|_\ell =$ $\boldsymbol{Q}$***.restrict***$(F|_{\rho-\{\ell\}}, V \cup \{\ell\})|_\ell =$ $\boldsymbol{Q}$***.restrict***$(F|_\rho, V)$ by Observation 3.3. To see that $V \cup \{\ell\}$ is downstream of $\rho - \{\ell\}$ we observe that ℓ is maximal in ρ, so it must be downstream of $\rho - \{\ell\}$. **For item 3**, let $\rho = \rho_1 \cup \rho_2 - \{\ell, \neg\ell\}$ and let V be $V_1 \cup V_2$.

[2] In the next section we will consider the case where F (the set of clauses) changes. However, the quantifier prefix, $\boldsymbol{Q}$, never changes so we can omit mentioning it in our notation.

We need to show that $\Gamma = \boldsymbol{Q}.\textit{restrict}(F|_\rho, V \cup \{\ell\})$ is true. Again we observe that ℓ appears in the outermost quantifier block among the variables in Γ. Thus Γ is true iff $\Gamma|_\ell$ and $\Gamma|_{\neg\ell}$ are both true. $\Gamma|_\ell = \boldsymbol{Q}.\textit{restrict}(F|_\rho, V \cup \{\ell\})|_\ell = \boldsymbol{Q}.\textit{restrict}(F|_{\rho\cup\{\ell\}}, V)$ (Observation 3.3). Then we have that $\boldsymbol{Q}.\textit{restrict}(F|_{\rho_1}, V_1)$ is true by assumption, that $\boldsymbol{Q}.\textit{restrict}(F|_{\rho_1}, V_1) \Rightarrow \boldsymbol{Q}.\textit{restrict}(F|_{\rho_1}, V)$ (since $V_1 \subseteq V$ and Observation 3.1), and that $\boldsymbol{Q}.\textit{restrict}(F|_{\rho_1}, V) \Rightarrow \boldsymbol{Q}.\textit{restrict}(F|_{\rho\cup\{\ell\}}, V)$ ($\rho \cup \{\ell\} = \rho_1 \cup (\rho_2 - \{\neg\ell\})$, all of the literals in $\rho_2 - \{\neg\ell\}$ are upstream of V, i.e., not in V, and thus Observation 3.4 applies). The proof for $\Gamma|_{\neg\ell}$ is similar. ■

Cubes are used to perform non-chronological solution backtracking that can skip large parts of the search space. They can also be stored and triggered to short-circuit the search of a subtree. In particular, if all of the literals in the cube are true, then the remaining theory is true and we need not descend in the search further.

Partial Cubes. With partitioning the leaf nodes satisfy only some of the clauses of F (see the sample path diagram at the start of this section). In particular, the clauses in the inactive partitions need not be satisfied by the assignments along the current path. Consider the operation of **QBF-Prt** where each invocation is a node in its search tree. Say that the search descends along a particular path arriving at node n_1 where the remaining theory partitions into Q_0, Q_1 and Q_2. We then choose to solve Q_0 (at line 2) in the next recursive call, and continue to descend reaching a node n_2 where the theory partitions again into P_1 and P_2. Continuing with P_1 we finally reach a leaf node n_ℓ without further splitting P_1.

At n_ℓ some subset of the original clauses F_1 have been made true by the literals set along the path to n_ℓ, and we can use item 1 of Theorem 1 to select a subset of these literals sufficient to form a cube for F_1: $\mathbf{cube}[\pi, \{\}, F_1]$. Note that in general this is not a cube for the original formula. In particular, we have not considered the clauses in the inactive partitions Q_1, Q_2 and P_2—these clauses have not necessarily been satisfied by the current path: $\mathbf{cube}[\pi, \{\}, F_1]$ is a partial cube. However, F_1 does include all clauses in the active partition P_1.

Now, we continue the search using this cube to backtrack to undo the most deeply assigned literal in π. If this literal is existential, we use item 2 of Theorem 1 to construct a new cube and backtrack further. If it is a universal we solve the other side, obtain another cube, combine the two cubes using item 3, and continue to backtrack further. At each node n we obtain a $\mathbf{cube}[\rho, V_1, F_1]$ such that ρ is a subset of the literals set along the path to n, F_1 includes all clauses in the active partition P_1 along with all other clauses made true along the path to n, and V_1 contains only variables instantiated below n (only variables backtracked over can be added into the cube). All of these variables are downstream of the variable instantiated at n which ensures that the conditions of item 3 of Theorem 1 are meet whenever it is to be applied. We also note that V_1 is always a subset of the variables of P_1 as these are the only variables branched on while solving P_1.

With partitioning, however, we cannot backtrack past node n_2 where the active partition P_1 was created—the remaining theory under n_2 is $P_1 \wedge P_2$ and we don't know yet if P_2 is true. Rather, when our search in the subtree solving P_1 finally produces a cube $[\rho_1, V_1, F_1]$ such that all of the literals of ρ_1 are true at or above n_2, we can backtrack to n_2 and then proceed to solve P_2.

If P_2 is true, the search in P_2's subtree will yield another cube, $\mathbf{cube}[\rho_2, V_2, F_2]$, such that F_2 includes all of the clauses of P_2 and shares with F_1 all clauses made true along the path to n_2, while V_2 is a subset of the variables of P_2. Now we want to combine these two cubes to learn a cube which will allow us to backtrack further within the subtree solving Q_0. We claim that $[\rho_1 \cup \rho_2, V_1 \cup V_2, F_1 \cup F_2]$ is the cube we want.

Theorem 2. *Given* $\mathbf{cube}[\rho_1, V_1, F_1]$ *and* $\mathbf{cube}[\rho_2, V_2, F_1]$ *such that (1)* $\rho_1 \cup \rho_2$ *is not contradictory (i.e.,* $\forall \ell \in \rho_1 \cup \rho_2 . \neg\ell \notin (\rho_1 \cup \rho_2)$*), (2) the variables in* $V_1 \cup V_2$ *are all downstream of the variables in* $\rho_1 \cup \rho_2$ *and (3)* V_1 *and* V_2 *share no existentials, then* $\mathbf{cube}[\rho_1 \cup \rho_2, V_1 \cup V_2, F_1 \cup F_2]$.

Proof: Since $V_1 \cup V_2$ is downstream of $\rho_1 \cup \rho_2$ by assumption we only need to prove that $\boldsymbol{Q}.\boldsymbol{restrict}(F_1 \cup F_2|_{\rho_1 \cup \rho_2}, V_1 \cup V_2)$ is true. For two QBF S_1 and S_2 we write $S_1 \Leftarrow S_2$ if S_2 true implies S_1 true.

$$
\begin{aligned}
& \boldsymbol{Q}.\boldsymbol{restrict}((F_1 \cup F_2)|_{\rho_1 \cup \rho_2}, V_1 \cup V_2) \\
\Leftarrow\ & \boldsymbol{Q}.\boldsymbol{restrict}((F_1 \wedge F_2)|_{\rho_1 \cup \rho_2}, V_1 \cup V_2) \\
\Leftarrow\ & \boldsymbol{Q}.\boldsymbol{restrict}(F_1|_{\rho_1 \cup \rho_2}, V_1 \cup V_2) \wedge \boldsymbol{restrict}(F_2|_{\rho_1 \cup \rho_2}, V_1 \cup V_2) \\
\Leftarrow\ & \boldsymbol{Q}.\boldsymbol{restrict}(F_1|_{\rho_1 \cup \rho_2}, V_1) \wedge \boldsymbol{restrict}(F_2|_{\rho_1 \cup \rho_2}, V_2) \\
\Leftarrow\ & \boldsymbol{Q}.\boldsymbol{restrict}(F_1|_{\rho_1 \cup \rho_2}, V_1) \wedge \boldsymbol{Q}.\boldsymbol{restrict}(F_2|_{\rho_1 \cup \rho_2}, V_2) \\
\Leftarrow\ & \boldsymbol{Q}.\boldsymbol{restrict}(F_1|_{\rho_1}, V_1) \wedge \boldsymbol{Q}.\boldsymbol{restrict}(F_2|_{\rho_2}, V_2)
\end{aligned}
$$

Line 1 might involve duplicating some clauses, but yields an equivalent formula. Line 2 is justified by the fact that both restriction and reduction are applied clause by clause. Line 3 is justified by Observation 3.1: we are restricting the clauses to a smaller set so the formula becomes stronger. Line 4 is justified because V_1 and V_2 share no existential variables so the formula can be partitioned. And finally line 5 is justified by Observation 3.4: none of the literals in $\rho_1 \cup \rho_2$ appear in $V_1 \cup V_2$. Finally, the conjunction on the last line is true by assumption. ■

This theorem says that once we obtain a cube for each partition P_1 and P_2 under the node n_2 we can form a cube that satisfies all of the clauses in P_1 and P_2 ($P_i \subseteq F_i$), as well as all of the clauses satisfied along the path to n_2. In otherwords, the new cube $\mathbf{cube}[\rho_1 \cup \rho_2, V_1 \cup V_2, F_1 \cup F_2]$ satisfies all of the clauses in the partition Q_0 and we can now utilize that cube to backtrack further within the subtree solving Q_0.

Note also that (1) all of the literals of ρ_i are contained in the path to n_2 thus $\rho_1 \cup \rho_2$ is not contradictory, (2) if v is the variable branched on at node n_2, then we have that all of the variables of V_i are downstream v and the literals in ρ_i are upstream of v thus $V_1 \cup V_2$ is downstream of $\rho_1 \cup \rho_2$, and (3) since P_1 and P_2 share no existentials neither do V_1 and V_2 since V_i is a subset of the variables in P_i.

Finally, we note that we can also trigger cubes $[\rho, V, F']$ by storing both ρ and V. In particular, it can be shown that it is sound to trigger a cube $[\rho, V, F']$ (and terminate the search of a subtree), if (a) all of the literals in ρ are true, (b) none of the existential variables of V are assigned, and (c) the existential variables of V form a partition at the current node of the search space. Note that we do not need to keep track of the clauses the cube covers F'. However, space precludes proving this result.

In sum, we have shown in this section how cubes can be used with partitioning for non-chronological solution backtracking, and that they can also be stored and triggered to short-circuit the search of a subtree.

4 Implementation

We have implemented dynamic partitioning within the DPLL based QBF solver 2clsQ [17,8]. In addition to the standard techniques employed in state of the art QBF solvers (e.g., solution analysis) 2clsQ also applies extensive binary clause reasoning at every search node [17]. However, 2clsQ also utilizes dynamic equality reduction which we turned off due to the logical and implementational difficulties that arise when applying equality reduction and partitioning simultaneously.

Partitions are computed at each decision level by a simple and straight forward depth-first search on the current theory. The complexity of this operation can be roughly stated as $O(|F| * vars_{\exists}(F))$ where $|F|$ denotes the size of the theory and $vars_{\exists}(F)$ the number of existentials in F.

We altered cube learning/solution backtracking as described in the previous section so that it could be used with dynamic partitioning. We also implemented a cube database and triggered cubes under the conditions described above.

The search requires a number of heuristic choices. Included in these choices are, deciding how to pick variables that are more likely to break the theory into partitions, deciding the order in which to solve detected partitions, and deciding when to turn off partition detection so as to minimize overhead.

A heuristic that selects a literal that satisfies the largest number of clauses can result in better partitioning since it decreases the overall connectivity. Computing articulation points in the corresponding graphical representation of the theory and branching accordingly is an alternate strategy for increasing partitioning. However, in our experiments it seemed that the best strategy was to branch on a literal that has the highest potential to cause a conflict, irrespective of its ability to generate partitions.

Similarly, there exist many ways to sort the computed partitions to decide which partition to process next. We used the following strategy: for each partition we computed the number of binary clauses that contain an active existentially quantified variable. Then we computed the ratio of active existentials and binary clauses in each partition further weighted by the number of active universals in the partition. This weighted ratio tries to capture the degree to which a partition is constrained. The lower the ratio the more constrainted are the existentials. The aim was to try to solve the most constrained partition first: if a partition failed we do not have to solve any of the other partitions as the conjunction is immediately false.

In our experiments we observed that partitioning can slow down the search process due to its high overhead. Computing partitions at each decision level is an expensive operation. Furthermore, it is wasted work if the theory consists of only one partition.

In general, it is unlikely that a theory breaks into multiple partitions when the ratio between clauses and existentially quantified variables is rather high (e.g., 15). And in fact empirically it turned out to be the case that when partitioning was turned off on instances with a high clause/variable ratio the resulting performance was consistently improved.

Furthermore, it seems to be the case that a theory with a rather low clause/variable ratio (e.g., 3) appears to be unsuitable for dynamic partitioning as well. In this case, the theory is easily solved without partitioning, so again partitioning is not worth the overhead. Hence, when the input instance has a low or high clause/variable ratio we do not bother to try to detect partitions, and simple solve the theory as if it is a single partition.

5 Experimental Results

To evaluate the empirical effect of our new approach we considered all of the non-random benchmark instances from QBFLib (2005) [18] (508 instances in total). We discarded the instances from the benchmark families von Neumann and Z since these can all be solved very quickly by any state of the art QBF solver (less than 10 sec. for the entire suite of instances). We also discarded the instances in benchmark families Uclid, Jmc, and Jmc-squaring. None of these instances can be solved within a time bound of 5,000 seconds by any of the QBF solvers we tested. This left us with 465 instances from 18 different benchmark families. We tested all of these instances on a Pentium 4 3.60GHz CPU with 6GB of memory (this is a 32 bit processor so only 4GB of this memory is actually addressable by our program). The time limit for a run of any solver was set to 5,000 seconds.

5.1 2clsQ vs. 2clsP

We first compared 2clsP with 2clsQ. These two solvers are the most similar, with 2clsP only adding partitioning to the processing already performed by 2clsQ (and subtracting equality reduction). Hence this comparison gives the most information on the effectiveness of partitioning taken in isolation. Table 1 shows the comparison between these two solvers. The table is broken down by benchmark family as the structural properties of the families can be quite distinct.

For each solver and benchmark the success rate and the time consumed by the solver on the successfully solved instances are displayed. Bold values indicate that the particular solver achieved the highest success rate on that families' instances, where ties are broken by CPU time consumed.

On this measure 2clsP is the best solver in 9 out of the 18 benchmark families. There exists only one benchmark family (*toilet*) where 2clsQ outperforms 2clsP. On the 8 remaining benchmark families 2clsP achieves the same performance as 2clsQ. On these benchmarks the clause/variable ratio was unfavorable for partitioning, so 2clsP operated without it on these families. That is, on these families 2clsP operates exactly the same as 2clsQ does. Normally, the clause/variable ratio stays fairly constant among the problems of the same benchmark family. However, in the case of the *toilet* benchmark the ratio varies across instances, so that some of the problems in this benchmark were solved by 2clsP using partitioning and others without. This also holds for other benchmarks (e.g., *Adder, S*).

The average success rate over all benchmark families is shown in the final row of the table. A high average displays fairly robust performance across structurally distinct

Table 1. Results achieved by 2clsQ and 2clsP on all tested benchmark families. Instances were timed out after 5,000 sec., and for each family the solver with highest success rate is shown in **bold**, where ties are broken by time required to solve these instances. The summary line shows the average success rate over all benchmark families and the total time taken (on solved instances only).

Benchmark Families (# instances)	*2clsQ*		*2clsP*	
	Succ. %	*time*	*Succ. %*	*time*
ADDER (16)	44%	5,267	**56%**	8,346
adder (16)	19%	0	**38%**	1,374
Blocks (16)	50%	46	50%	46
C (24)	21%	16	**25%**	14
Connect (60)	100%	66	100%	66
Counter (24)	33%	4,319	**33%**	1,220
EV-Pursuer(38)	26%	2,836	**34%**	2,282
FlipFlop (10)	100%	4	100%	4
K (107)	35%	20,575	**36%**	20,039
Lut (5)	100%	19	100%	19
Mutex (7)	43%	22	43%	22
Qshifter (6)	33%	59	**67%**	1,924
S (52)	8%	9	**15%**	3,405
Szymanski (12)	67%	2,741	67%	2,741
TOILET (8)	75%	528	75%	528
toilet (38)	**84%**	47	84%	531
Tree (14)	100%	296	**100%**	0
Summary	58%	36,791	63%	42,502

instances. On this measure 2clsP is superior to 2clsQ solving 63% of all instances on average compared to 58%.

The total CPU time is lower with 2clsQ than with 2clsP which was expected. Computing partitions at every decision level is an expensive operation. In summary, these results demonstrate quite convincingly that our new technique offers robust improvements to 2clsQ.

5.2 2clsP vs. Other Solvers

We also compared our new solver 2clsP to five other state of the art QBF solvers **Quaffle** [13] (version as of Feb. 2005), **Quantor** [9] (version as of 2004), **Qube** (release 1.3) [19], **Skizzo** [10] (release 0.82), **SQBF** [20].

Quaffle, Qube, and SQBF are based on search, whereas Quantor is based on variable elimination and SAT grounding. Skizzo uses a combination of variable elimination, SAT grounding, and search, and also applies a variety of other kinds of reasoning on the symbolic and the ground representations of the instances.

Table 2 shows the performance of 2clsP and all other search based solvers on the 465 problem instances we tested, broken down by benchmark family.

Table 2. Results achieved by 2clsP and five other state-of-the-art QBF solvers on all tested benchmark families. Unsolved instances were timed out after 5,000 sec., and for each family the solver with highest success rate is shown in **bold**, where ties are broken by time required to solve these instances. The summary line shows the average success rate over all benchmark families and the total time taken (on solved instances only).

Benchmark Families	*Skizzo*		*Quantor*		*2clsP*		*Quaffle*		*Qube*		*SQBF*	
(# instances)	*Succ. %*	*time*	*Succ. %*	*time*	*Succ. %*	*time*	*Succ. %*	*time*	*Succ. %*	*time*	*Succ. %*	*time*
ADDER (16)	50%	954	25%	24	**56%**	8,346	25%	1	13%	72	13%	3
adder (16)	44%	455	25%	29	38%	1,374	42%	5	**44%**	0	38%	2,678
Blocks (16)	56%	108	**100%**	308	50%	46	75%	1,284	69%	1774	75%	7,042
C (24)	25%	1,070	21%	140	**25%**	14	21%	5,356	8%	3	17%	4
Chain (12)	100%	1	**100%**	0	**100%**	0	67%	6,075	83%	4,990	58%	4,192
Connect (60)	68%	802	67%	14	**100%**	7	70%	253	75%	7,013	67%	0
Counter (24)	**54%**	1,036	50%	217	33%	1,220	38%	5	33%	2	38%	9
EVPursade (38)	29%	1,450	3%	73	**34%**	2,282	26%	1,962	18%	4,402	32%	4,759
FlipFlop (10)	100%	6	100%	3	100%	4	**100%**	0	100%	1	80%	5,027
K (107)	**88%**	1,972	63%	3,839	36%	20,039	35%	21,675	37%	21,801	33%	5,563
Lut (5)	100%	9	**100%**	3	100%	19	100%	1	**100%**	3	100%	1,247
Mutex (7)	100%	0	43%	0	43%	22	29%	43	43%	64	43%	1
Qshifter (6)	**100%**	8	100%	26	67%	1,924	17%	0	33%	29	33%	1,107
S (52)	**27%**	644	25%	910	15%	3,405	2%	0	4%	401	2%	0
Szymanski (12)	42%	1,147	25%	7	**67%**	2,741	0%	0	8%	0	0%	0
TOILET (8)	**100%**	1	100%	4,135	75%	528	75%	61	63%	496	100%	1,307
toilet (38)	100%	84	100%	684	84%	531	97%	115	**100%**	58	97%	395
Tree (14)	**100%**	0	**100%**	0	**100%**	0	100%	37	**100%**	0	93%	1,051
Summary	71%	9,747	64%	10,412	63%	42,502	51%	36,873	52%	41,109	51%	34,385

As in the previous table we display for each solver and benchmark the success rate and the time consumed by the solver on the successfully solved instances. Again, bold values indicate that the particular solver gained the highest success rate on that families' instances breaking ties by CPU time consumed.

On this measure 2clsP is the best solver on 7 out of the 18 benchmark families. Skizzo follows with 6, Quantor with 4, Qube with 4, and Quaffle with 1. SQBF is not the best performer on any benchmark family.

The average success rate over all benchmark families is shown in the final row of the table. A high average displays fairly robust performance across structurally distinct instances. On this measure 2clsP is superior to all other search based solvers with an average success rate of 63%. It is followed by Qube (-11%), SQBF (-12%) and Quaffle (-12%). However, both Skizzo ($+8\%$) and Quantor ($+1\%$) achieve a better average success rate. In terms of the total CPU time, 2clsP requires the highest amount of CPU time.

In total 2clsP is a very competitive QBF solver that achieves the best performance on more benchmark families than any other solver. In addition, its average success rate is close to the best achieved by any of the tested solvers. Although the new techniques

employed in 2clsP are rather complex we see that they pay off in terms of performance gains.

5.3 State of the Art Solver

The results of the QBF competition 2006 [21] indicate that the "best" QBF solver would probably use a portfolio approach rather than any single solver. For example, our 2clsQ entry which won the 2006 competition first applied a hyperbinary preprocessor (PreQuel [11,22]), then it ran the QBF solver Quantor for a fixed period of time. Finally if the problem was still not solved 2clsQ was invoked on output of PreQuel.

Given the results displayed in [11] a very promising strategy in the competition would be to apply PreQuel and a time-limited version of Quantor as before, and Skizzo as final solver. This observation is mainly due to the good standard performance of Skizzo and the positive impact of preprocessing on Skizzo [11]. It is not clear if the employment of Quantor in the context of Skizzo is as beneficial as it is for a search-based solver but given the performance of Quantor it should not turn out to be a drawback either.

However, depending on the benchmark families in the competition, the results shown here indicate that 2clsP together with the initial two stage processing of PreQuel and Quantor would also be able to achieve a high ranking. This is due to the fact that 2clsP remains to be a competitive solver on several benchmark families even when Skizzo is supplied with a preprocessed problem instances (e.g., the *Adder* benchmark family).

6 Conclusions

We have shown how dynamic partitioning can be used to obtain significant improvements to a state of the art QBF solver, 2clsQ. The key to making dynamic partitioning work is finding a way to utilize clause and cube learning in conjunction with partitioning. In this paper we have presented an approach for accomplishing this.

There is, however, much scope for further improvements. These include better heuristics for promoting the dynamic creation of partitions, and better heuristics for deciding when to and when not to partition. Also the theory behind partial cubes can probably be elaborated further and perhaps used to obtain further algorithmic insights.

References

1. Bryant, R., Lahiri, S., Seshia, S.: Convergence testing in term-level bounded model checking. Technical Report CMU-CS-03-156, Carnegie Mellon University (2003)
2. Egly, U., Eiter, T., Tompits, H., Woltran, S.: Solving advanced reasoning tasks using quantified boolean formulas. In: AAAI/IAAI. (2000) 417–422
3. Rintanen, J.: Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research **10** (1999) 323–352
4. Ali, M., Safarpour, S., Veneris, A., Abadir, M., Drechsler, R.: Post-verification debugging of hierarchical designs. In: International Conf. on Computer Aided Design (ICCAD). (2005) 871–876

5. Jr., R.J.B., Pehoushek, J.D.: Counting models using connected components. In: Proceedings of the AAAI National Conference (AAAI). (2000) 157–162
6. Darwiche, A.: On the tractable counting of theory models and its application to truth maintenance and belief revision. Journal of Applied Non-Classical Logics **11**(1-2) (2001) 11–34
7. Bacchus, F., Dalmao, S., Pitassi, T.: Algorithms and Complexity Results for #SAT and Bayesian Inference. In: 44th Symposium on Foundations of Computer Science (FOCS). (2003) 340–351
8. Samulowitz, H., Bacchus, F.: QBF Solver 2clsQ (2006) available at http://www.cs.toronto.edu/~fbacchus/sat.html
9. Biere, A.: Resolve and expand. In: Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT). (2004) 238–246
10. Benedetti, M.: sKizzo: a QBF decision procedure based on propositional skolemization and symbolic reasoning. Technical Report TR04-11-03 (2004)
11. Samulowitz, H., Davies, J., Bacchus, F.: Preprocessing QBF. In: Principles and Practice of Constraint Programming, Springer-Verlag, New York (2006)
12. Büning, H.K., Karpinski, M., Flügel, A.: Resolution for quantified boolean formulas. Inf. Comput. **117**(1) (1995) 12–18
13. Zhang, L., Malik, S.: Towards symmetric treatment of conflicts and satisfaction in quantified boolean satisfiability solver. In: Principles and Practice of Constraint Programming (CP2002). (2002) 185–199
14. Giunchiglia, E., Narizzano, M., Tacchella, A.: Learning for quantified boolean logic satisfiability. In: Eighteenth national conference on Artificial intelligence. (2002) 649–654
15. Benedetti, M.: Quantifier Trees for QBFs. In: Proc. of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT05). (2005)
16. Sang, T., Bacchus, F., Beame, P., Kautz, H., Pitassi, T.: Combining component caching and clause learning for effective model counting. In: SAT. (2004)
17. Samulowitz, H., Bacchus, F.: Binary clause reasoning in qbf. In: Ninth International Conference on Theory and Applications of Satisfiability Testing (SAT 2006), Lecture Notes in Computer Science 2919. (2006)
18. Giunchiglia, E., Narizzano, M., Tacchella, A.: Quantified Boolean Formulas satisfiability library (QBFLIB) (2001) `www.qbflib.org`.
19. Giunchiglia, E., Narizzano, M., Tacchella, A.: QUBE: A system for deciding quantified boolean formulas satisfiability. In: International Joint Conference on Automated Reasoning (IJCAR). (2001) 364–369
20. Samulowitz, H., Bacchus, F.: Using SAT in QBF. In: Principles and Practice of Constraint Programming, Springer-Verlag, New York (2005)
21. Giunchiglia, E., Narizzano, M., Tacchella, A.: The qbf2006 competition (2006) available on line at http://www.qbflib.org/.
22. Samulowitz, H., Davies, J., Bacchus, F.: QBF Preprocessor Prequel (2006) available at http://www.cs.toronto.edu/~fbacchus/sat.html.

Backdoor Sets of Quantified Boolean Formulas*

Marko Samer and Stefan Szeider

Department of Computer Science
Durham University, UK
{marko.samer,stefan.szeider}@durham.ac.uk

Abstract. We generalize the notion of backdoor sets from propositional formulas to quantified Boolean formulas in conjunctive normal form (QCNF). We develop parameterized algorithms that admit uniform polynomial time QCNF evaluation parameterized by the size of smallest strong backdoor sets. For our algorithms we develop a theory of variable dependency which is of independent interest. As a result, we obtain hierarchies of classes of tractable QCNF formulas with the classes of quantified Horn and quantified 2CNF formulas, respectively, at their first level, thus gradually generalizing these two prominent tractable classes. In contrast to known tractable classes based on bounded treewidth, the number of quantifier alternations of our classes is unbounded.

1 Introduction

Several important computational tasks like planning, verification, and several questions of automated reasoning and games can be naturally encoded as the evaluation problem of *quantified Boolean formulas* [17,20,22], a generalization of the propositional satisfiability problem (SAT). In recent years quantified Boolean formulas have become a very active research area. The evaluation of quantified Boolean formulas constitutes a PSPACE-complete problem [24] and is therefore computationally harder than the NP-complete propositional satisfiability problem. For background information on quantified Boolean formulas we refer the reader to other sources [11,19]. In the sequel we make the common assumption that for a given formula all variables are quantified (i.e., there are no free variables) and that the formula is in prenex normal form with the propositional part (the matrix) in conjunctive normal form; we will refer to such formulas as *QCNF formulas*.

Every propositional CNF formula can be considered as a QCNF formula with all variables existentially quantified. Thus, every tractable class of CNF formulas (dozens of such classes are known) gives rise to a tractable class of QCNF formulas. However, very few tractable classes of quantified Boolean formulas are known where the number of *quantifier alternations* is unbounded. For example, the time needed for solving QCNF formulas of bounded treewidth grows non-elementary in the number of quantifier alternations, as recently shown by Pan and Vardi [18]. Two prominent tractable classes with *unbounded* quantifier alternations are QHORN (clauses contain at most one positive literal) and Q2CNF (clauses contain at most two literals). QHORN formulas and Q2CNF formulas can be evaluated in polynomial time due to classic results

* Research supported by the EPSRC project EP/E001394/1.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 230–243, 2007.

of Kleine Büning, Karpinski, and Flögel [10] and of Aspvall, Plass, and Tarjan [1], respectively.

In this paper we define hierarchies of tractable classes of QCNF formulas of the form $\mathcal{C}_0 \subseteq \mathcal{C}_1 \subseteq \mathcal{C}_2 \subseteq \cdots$ where the first class $\mathcal{C}_0$ is either QHORN or Q2CNF, and every QCNF formula belongs to some $\mathcal{C}_k$ for k large enough. We develop algorithms which render membership in $\mathcal{C}_k$ as well as evaluation of formulas in $\mathcal{C}_k$ feasible in *uniform polynomial time*. In other words, our algorithms are *fixed-parameter algorithms* since the order of the polynomial that bounds their running time is independent of k. This feature of fixed-parameter algorithms admits an efficient processing of large instances as long as the parameter k is kept small (in contrast to non-uniform polynomial-time algorithms with a running time of, say, $\mathcal{O}(n^k)$). We will briefly review some basic concepts of fixed-parameter complexity in Section 2.2.

Backdoor Sets

Our approach is based on the generalization of the concept of backdoor sets from propositional satisfiability to quantified Boolean formulas. Backdoor sets for SAT (and similarly for constraint satisfaction) were introduced by Williams, Gomes, and Selman as a tool for analyzing the performance of local search algorithms [27,28]. Backdoor sets have recently received a lot of attention in satisfiability research [7,8,9,13,15,16,21,26]. The idea is to consider a *base class* $\mathcal{C}$ of CNF formulas for which membership and satisfiability are both decidable in polynomial time. A set B of variables of an arbitrary CNF formula F is a *strong $\mathcal{C}$-backdoor set* if all formulas that can be obtained from F by instantiating the variables in B belong to the base class $\mathcal{C}$ (in the sequel we will also discuss the notion of *weak backdoor sets* which is, however, less relevant for our considerations). If a strong backdoor set B is known, we can efficiently decide the satisfiability of F by checking the satisfiability of $2^{|B|}$ CNF formulas that belong to the tractable class $\mathcal{C}$. Nishimura, Ragde, and Szeider have studied the algorithmic complexity of finding small strong backdoor sets of CNF formulas with respect to various base classes including Horn and 2CNF formulas [15], formulas that can be decided by polynomial-time DLL subsolvers [25], and variable-disjoint unions of hitting formulas (clustering formulas) [16].

In this paper we generalize the concepts of weak and strong backdoor sets to QCNF formulas. In the following we discuss some basic principles of our approach, taking the class of (quantified) Horn formulas as the base class. Consider the CNF formula

$$F = (\neg x \vee y \vee \neg w) \wedge (x \vee \neg y \vee w) \wedge (\neg y \vee z) \wedge (y \vee \neg z).$$

The set $B = \{x\}$ is a strong HORN-backdoor set of F since for $x = 0$ we obtain the clauses $(\neg y \vee w)$, $(\neg y \vee z)$, and $(y \vee \neg z)$ and for $x = 1$ we obtain the clauses $(y \vee \neg w)$, $(\neg y \vee z)$, and $(y \vee \neg z)$ which are all Horn. Now let us quantify the variables so that we obtain the QCNF formula

$$\mathcal{F} = \forall y \, \forall z \, \exists x \, \exists w \, F.$$

Obviously, the variable x cannot be isolated anymore in a backdoor set as above since the truth value of x apparently depends on the truth values of y and z. In other words, we cannot reduce the evaluation of $\mathcal{F}$ to the evaluation of some simpler QHORN formula

obtained by fixing the truth value of x while y and z remain universally quantified. Hence, for QCNF formulas we require that strong backdoor sets are closed with respect to the dependency of variables: if x belongs to the backdoor set B, also all variables on which x depends belong to B.

So far we have left open the exact meaning of "x depends on y." Clearly it would be safe to assume that a variable x depends on all variables that are quantified left of x. Thus, in the above example, $\{x, y, z\}$ certainly constitutes a strong QHORN-backdoor set of $\mathcal{F}$. However, a closer look at the formula reveals that we can do better. Although z is quantified left of x we can actually swap the quantification of x and z, revealing that x does not depend on z. Namely, the matrix F can be split into two parts $F_1 = (\neg x \vee y \vee \neg w) \wedge (x \vee \neg y \vee w)$ and $F_2 = (\neg y \vee z) \wedge (y \vee \neg z)$ such that x and w occur only in F_1 and z occurs only in F_2. Thus, we can rewrite $\mathcal{F}$ equivalently as

$$\forall y\, \forall z\, \exists x\, \exists w\, (F_1 \wedge F_2) \;\Leftrightarrow\; \forall y\, \big((\exists x\, \exists w\, F_1) \wedge (\forall z\, F_2)\big) \;\Leftrightarrow\; \forall y\, \exists x\, \exists w\, \forall z\, (F_1 \wedge F_2),$$

thus shifting x to the left and so showing that x does not depend on z. Consequently, we can actually form the smaller backdoor set $\{x, y\}$. With a more sophisticated reasoning that we will describe in Section 3, we can shift x to the left of z even if x occurs in F_2, as long as it occurs only positively or only negatively. For the general case we need to take into account whether variables are connected to each other in a certain way.

Along these lines we develop a *scheme of variable dependency* that allows us to limit the blow-up of strong backdoor sets caused by variable dependencies. Variable dependency has been studied in a slightly different context by Egly, Tompits, and Woltran [5] and by Biere [3]; of related interest is Benedetti's work on quantifier trees [2]. For a variable dependency scheme one needs to compromise between feasibility and generality: we show in Section 3 that identifying minimal variable dependencies is PSPACE-hard. We propose a dependency scheme that is reasonably general and subsumes the scheme that arises by the methods of Egly et al. [5] and Biere [3]. We formulate our scheme strictly in terms of QCNF formulas, allowing a direct implementation within the data structures used by QCNF-based solvers. The application of our dependency scheme is not limited to backdoor set optimization; we think that it is also useful for other aspects of the evaluation of quantified Boolean formulas.

Results

We develop algorithms that find strong backdoor sets with respect to the base classes QHORN and Q2CNF, taking into account any tractable variable dependency scheme. The algorithms detect strong QHORN-backdoor sets and strong Q2CNF-backdoor sets of size bounded by a constant in linear time, assuming that the variable dependencies are provided as an input. Similarly as the algorithms suggested by Nishimura et al. [15], our algorithms are based on the bounded search tree technique, a fundamental technique for fixed-parameter algorithms [4,14]. Once a strong backdoor set is found, the formula can be evaluated by considering all truth assignments to the variables in the backdoor set. Thus, if we take $\mathcal{C}_k$ as the class of QCNF formulas that have strong QHORN-backdoor sets (respectively strong Q2CNF-backdoor sets) of size at most k, then we have indeed an infinite hierarchy of tractable classes of QCNF formulas with the base class QHORN

(respectively Q2CNF) at its first level. Every QCNF formula belongs to some $\mathcal{C}_k$ for k large enough, and every class $\mathcal{C}_k$ contains formulas with arbitrarily many quantifier alternations.

Thus, we have fixed-parameter tractability results for a problem that is PSPACE-hard in the non-parameterized sense. Here, the gain due to parameterization is even more drastic than it is for most of the known fixed-parameter tractability results where the non-parameterized problems are "only" NP-complete.

2 Background

2.1 Quantified Boolean Formulas

We consider propositional formulas F in conjunctive normal form (CNF). We identify each CNF formula with the set of its clauses, e.g., $F = (\neg x \vee y \vee z) \wedge (\neg y \vee \neg z) \wedge (x \vee \neg y)$ is identified with the set $\{\{\neg x, y, z\}, \{\neg y, \neg z\}, \{x, \neg y\}\}$. Moreover, we consider quantified Boolean formulas in quantified CNF (QCNF), for example,

$$\mathcal{F} = \forall x\, \exists y\, \forall z\, F = \forall x\, \exists y\, \forall z\, (\neg x \vee y \vee z) \wedge (\neg y \vee \neg z) \wedge (x \vee \neg y).$$

We refer to F as the *matrix* of $\mathcal{F}$. We assume that all variables occurring in the matrix are bounded by some quantifier, i.e., there are no free variables in $\mathcal{F}$, and that all variables bounded by some quantifier occur in the matrix. Each clause in F is a finite set of *literals*, and a literal is a negated or unnegated propositional *variable*. For a literal ℓ we denote by $\overline{\ell}$ the literal of opposite polarity, i.e., $\overline{x} = \neg x$ and $\overline{\neg x} = x$; moreover, for a set X of literals, we put $\overline{X} = \{\, \overline{\ell} : \ell \in X \,\}$. For a clause C we denote by $var(C)$ the set of variables that occur (negated or unnegated) in C. For a QCNF formula $\mathcal{F}$ and its matrix F we put $var(\mathcal{F}) = var(F) = \bigcup_{C \in F} var(C)$.

For a CNF formula F and a variable $x \in var(F)$, we put $F - x = \{\, C \setminus \{x, \overline{x}\} : C \in F \,\}$; moreover, for a set $X \subseteq var(F)$, we put $F - X = \{\, C \setminus (X \cup \overline{X}) : C \in F \,\}$. For a QCNF formula $\mathcal{F} = \mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n\, F$ and a variable $x_p \in var(\mathcal{F})$, we denote by $\mathcal{F} - x_p$ the QCNF formula obtained from $\mathcal{F}$ by replacing the matrix F by $F - x_p$ and removing the superfluous quantification $\mathsf{Q}_p x_p$; moreover, we generalize this notation in a straight-forward way to $\mathcal{F} - X$ for sets $X \subseteq var(\mathcal{F})$. We define the *depth* of x_p in $\mathcal{F}$ as $\delta_{\mathcal{F}}(x_p) = p$ and we put $q_{\mathcal{F}}(x_p) = \mathsf{Q}_p$. A QCNF formula $\mathcal{F}'$ is obtained from $\mathcal{F}$ by *quantifier reordering*, if there is a permutation $i_1, \ldots, i_n$ of $1, \ldots, n$ such that $\mathcal{F}' = \mathsf{Q}_{i_1} x_{i_1} \ldots \mathsf{Q}_{i_n} x_{i_n}\, F$.

A *truth assignment* is a mapping $\tau : X \to \{0, 1\}$ defined on some set X of variables. We extend τ to literals by setting $\tau(\neg x) = 1 - \tau(x)$ for $x \in X$. For a truth assignment $\tau : \{x\} \to \{0, 1\}$ we simply write "$x = 0$" and "$x = 1$" respectively. For a truth assignment τ and a CNF formula F, we denote by $F[\tau]$ the CNF formula obtained from F by removing all clauses which contain a literal ℓ with $\tau(\ell) = 1$ and by removing literals ℓ with $\tau(\ell) = 0$ from the remaining clauses; moreover, for a QCNF formula $\mathcal{F} = \mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n\, F$, we denote by $\mathcal{F}[\tau]$ the QCNF formula obtained from $\mathcal{F}$ by replacing the matrix F by $F[\tau]$ and removing all superfluous quantifications. A truth assignment τ *satisfies* a CNF formula F if $F[\tau] = \emptyset$.

The evaluation function $\nu : \mathcal{F} \mapsto \{0, 1\}$ on QCNF formulas $\mathcal{F}$ is recursively defined by $\nu(\exists x\, \mathcal{F}) = \max(\nu(\mathcal{F}[x = 0]), \nu(\mathcal{F}[x = 1]))$, $\nu(\forall x\, \mathcal{F}) = \min(\nu(\mathcal{F}[x = 0]),$

$\nu(\mathcal{F}[x = 1]))$, and, if $\mathcal{F}$ has no variables, $\nu(\mathcal{F}) = 1$ if $\mathcal{F} = \emptyset$ and $\nu(\mathcal{F}) = 0$ otherwise. A QCNF formula $\mathcal{F}$ is *true* (or *satisfiable*) if $\nu(\mathcal{F}) = 1$; otherwise it is *false* (or *unsatisfiable*). Two QCNF formulas $\mathcal{F}$ and $\mathcal{F}'$ are *equivalent* if $\nu(\mathcal{F}) = \nu(\mathcal{F}')$.

A clause is called *Horn* if it contains at most one positive literal and it is called *binary* if it contains at most two literals. A CNF/QCNF formula is called Horn (resp. binary) if all its clauses are Horn (resp. binary). The class of Horn (resp. binary) CNF formulas is denoted by HORN (resp. 2CNF); the class of Horn (resp. binary) QCNF formulas is denoted by QHORN (resp. Q2CNF).

2.2 Parameterized Complexity

An instance of a parameterized problem is a pair (I, k) where I is the *main part* and k is the *parameter*; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* if it can be solved by a fixed-parameter algorithm, i.e., if instances (I, k) can be solved in time $\mathcal{O}(f(k)\, n^c)$, where f is a computable function, c is a constant, and n is the size of I. FPT denotes the class of all fixed-parameter tractable decision problems [4,6,14].

Parameterized complexity offers a *completeness theory*, similar to the theory of NP-completeness, that allows the accumulation of strong theoretical evidence that a parameterized problem is *not* fixed-parameter tractable. This completeness theory is based on the *weft hierarchy* of complexity classes W[t], $t \geq 1$. Each class is the equivalence class of certain parameterized satisfiability problems under parameterized reductions ("fpt-reductions") which are straightforward extensions of polynomial-time many-to-one reductions that ensure a parameter of one problem maps into a parameter of another problem [4,6,14]. If we know that a parameterized problem is W[t]-hard (under parameterized reductions) for some $t \geq 1$, then it is very unlikely that the problem is fixed-parameter tractable. Fixed-parameter tractability of the problem would imply that the Exponential Time Hypothesis fails [6] (i.e., the existence of a $2^{o(n)}$ algorithm for n-variable 3SAT).

3 Dependency Schemes

As already mentioned in the introduction, we consider *dependency schemes* in order to obtain smaller backdoor sets. To this aim, we first need the following notions: L and R assign to each QCNF formula $\mathcal{F}$ and $x \in var(\mathcal{F})$ the sets $L_\mathcal{F}(x) = \{\, y \in var(\mathcal{F}) : 1 \leq \delta_\mathcal{F}(y) \leq \delta_\mathcal{F}(x) \,\}$ and $R_\mathcal{F}(x) = \{\, y \in var(\mathcal{F}) : \delta_\mathcal{F}(x) \leq \delta_\mathcal{F}(y) \leq |var(\mathcal{F})| \,\}$.

Definition 1 (Up-shifting). *Let $\mathcal{F}$ be a QCNF formula and $D \subseteq var(\mathcal{F})$. We say the QCNF formula $\mathcal{F}'$ is obtained from $\mathcal{F}$ by* up-shifting *D, in symbols $\mathcal{F}' = S^{\uparrow}_D(\mathcal{F})$, if $\mathcal{F}'$ is obtained from $\mathcal{F}$ by quantifier reordering and the following holds:*

1. *$L_{\mathcal{F}'}(x) = D$ for some $x \in var(\mathcal{F}) = var(\mathcal{F}')$ and*
2. *$\delta_{\mathcal{F}'}(x) < \delta_{\mathcal{F}'}(y)$ if and only if $\delta_\mathcal{F}(x) < \delta_\mathcal{F}(y)$ for all $x, y \in D$ and*
3. *$\delta_{\mathcal{F}'}(x) < \delta_{\mathcal{F}'}(y)$ if and only if $\delta_\mathcal{F}(x) < \delta_\mathcal{F}(y)$ for all $x, y \in var(\mathcal{F}) \setminus D$.*

For example, recall the QCNF formula $\mathcal{F} = \forall y \forall z \exists x \exists w\, F$ from the introduction and let $D = \{x, y\}$. Then we have $\mathcal{F}' = S^{\uparrow}_D(\mathcal{F}) = \forall y \exists x \forall z \exists w\, F$. Note that up-shifting does not preserve equivalence in general.

Definition 2 (Dependency scheme). *A* dependency scheme D *assigns to each QCNF formula* $\mathcal{F}$ *and variable* $x \in var(\mathcal{F})$ *a set* $D_{\mathcal{F}}(x) \subseteq var(\mathcal{F})$ *such that* $\mathcal{F}$ *and* $S^{\uparrow}_{D_{\mathcal{F}}(x)}(\mathcal{F})$ *are equivalent. A dependency scheme* D *is* tractable *if* $D_{\mathcal{F}}(x)$ *can be computed in time that is polynomial in* $\mathcal{F}$.

For a dependency scheme D and $X \subseteq var(\mathcal{F})$, we put $D_{\mathcal{F}}(X) = \bigcup_{x \in X} D_{\mathcal{F}}(x)$.

This is justified by the following lemma which follows by induction on $|X|$.

Lemma 1. *Let* $\mathcal{F}$ *be a QCNF formula and* D *be a dependency scheme. Moreover, let* $X \subseteq var(\mathcal{F})$ *and* $Y = \bigcup_{x \in X} D_{\mathcal{F}}(x)$. *Then* $\mathcal{F}$ *and* $S^{\uparrow}_{Y}(\mathcal{F})$ *are equivalent.*

A simple example of a tractable dependency scheme is L as defined above. We call L the *trivial dependency scheme* since always $\mathcal{F} = S^{\uparrow}_{L_{\mathcal{F}}(x)}(\mathcal{F})$.

Our aim in the following is to find tractable dependency schemes such that the sets $D_{\mathcal{F}}(x)$ are as small as possible. We say that dependency scheme D is *more general* than dependency scheme D' if always $D_{\mathcal{F}}(x) \subseteq D'_{\mathcal{F}}(x)$ and if the inclusion is strict in some cases. The following dependency scheme is tractable and more general than L:

The *standard dependency scheme* D^{std} is based on standard quantifier shifting rules as considered by Egly et al. [5]. In short, D^{std} can be defined by assigning to each QCNF formula $\mathcal{F}$ and $x \in var(\mathcal{F})$ the set $D^{\text{std}}_{\mathcal{F}}(x) \subseteq L_{\mathcal{F}}(x)$ consisting of x and all variables that appear in front of x after shifting the quantifiers down the parse tree of the formula as far as possible according to quantifier shifting rules. Biere [3] considers a very similar notion of variable dependency. Note that the standard dependency scheme is not as general as possible. The following proposition shows that when we want a dependency scheme to be tractable, we cannot expect it to be optimal.

Proposition 1. *Let* $\mathcal{F}$ *be a QCNF formula and* $x, y \in var(\mathcal{F})$. *The problem of deciding whether there exists a dependency scheme* D *such that* $y \notin D_{\mathcal{F}}(x)$ *is* PSPACE*-complete.*

Proof. (Sketch.) The problem belongs to PSPACE as polynomial space suffices to go through all QCNF formulas $\mathcal{F}'$ obtained from $\mathcal{F}$ by quantifier reordering such that $y \notin L_{\mathcal{F}'}(x)$ and to check whether $\mathcal{F}$ and $\mathcal{F}'$ are equivalent. For showing PSPACE-hardness, we reduce from QBF satisfiability. To this aim, let $\mathcal{G} = \mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n\, G$ be an arbitrary QCNF formula and $y, z \notin var(\mathcal{G})$ be two new variables. We put $F = G \wedge (y \vee z) \wedge (\neg y \vee \neg z)$ and $\mathcal{F} = \forall y \exists z\, \mathsf{Q}_1 x_1 \ldots \mathsf{Q}_n x_n\, F$. It follows then that $\mathcal{F}$ and $S^{\uparrow}_{D_{\mathcal{F}}(z)}(\mathcal{F})$ are equivalent if and only if $\mathcal{G}$ is false. □

Next we define a tractable dependency scheme that improves upon the standard dependency scheme, providing a fair compromise between tractability and optimality.

Definition 3 (Dependency triangle). *Let* $\mathcal{F}$ *be a QCNF formula with matrix* F. *Two clauses* $C, C' \in F$ *are* connected *with respect to* $X \subseteq var(\mathcal{F})$ *if there is a sequence* $C_1, \ldots, C_n$ *with* $C = C_1$ *and* $C' = C_n$, *such that* $var(C_i) \cap var(C_{i+1}) \cap X \neq \emptyset$ *for all* $1 \leq i < n$. *Three clauses* $C_1, C_2, C_3 \in F$ *form an* (x, y)-dependency triangle *with respect to* $X \subseteq var(\mathcal{F})$ *if (i)* $q_{\mathcal{F}}(x) = \forall$ *and* $q_{\mathcal{F}}(y) = \exists$, *(ii)* C_1 *and* C_2 *as well as* C_1 *and* C_3 *are connected with respect to* $X \cup \{x\}$, *and (iii)* $x \in var(C_1)$, $y \in C_2$, *and* $\neg y \in C_3$.

To illustrate these definitions, consider the QCNF formula $\mathcal{F} = \forall u\,\exists v\,\exists w\,\forall x\,\exists y\,\forall z\,(u \vee y) \wedge (x \vee y) \wedge (\neg v \vee w \vee x) \wedge (v \vee w \vee \neg z) \wedge (v \vee \neg y) \wedge (\neg w \vee z)$. Each pair of clauses in $\mathcal{F}$ is connected with respect to some set. For example, the clauses $(x \vee y)$ and $(v \vee w \vee \neg z)$ are connected with respect to $\{v, y\}$ and with respect to $\{w, x\}$. Furthermore, there is a (u, v)-dependency triangle with respect to $\{x, y\}$ (by choosing $C_1 = u \vee y$, $C_2 = v \vee \neg y$, and $C_3 = \neg v \vee w \vee x$) and a (z, w)-dependency triangle with respect to $\emptyset$ (by choosing $C_1 = C_2 = v \vee w \vee \neg z$ and $C_3 = \neg w \vee z$).

Definition 4 (Triangle dependency scheme). *The* triangle dependency scheme $D^{\triangle}$ *assigns to each QCNF formula $\mathcal{F}$ and $x \in var(\mathcal{F})$ the set $D^{\triangle}_{\mathcal{F}}(x) = D^{\delta_{\mathcal{F}}(x)-1}_{\mathcal{F}}(x)$, where $D^{\delta_{\mathcal{F}}(x)-1}_{\mathcal{F}}(x) \subseteq L_{\mathcal{F}}(x) \subseteq var(\mathcal{F})$ is recursively defined as follows: (i) $D^0_{\mathcal{F}}(x) = \{x\}$, (ii) $D^{i+1}_{\mathcal{F}}(x) = D^i_{\mathcal{F}}(x) \cup \{y\}$ if $y \in var(\mathcal{F})$ such that $\delta_{\mathcal{F}}(y) = \delta_{\mathcal{F}}(x) - i$ and one of the following two conditions holds, and (iii) $D^{i+1}_{\mathcal{F}}(x) = D^i_{\mathcal{F}}(x)$ otherwise.*

1. *There exists an (x, y)-dependency triangle (i.e., $q_{\mathcal{F}}(x) = \forall$ and $q_{\mathcal{F}}(y) = \exists$) or a (y, x)-dependency triangle (i.e., $q_{\mathcal{F}}(y) = \forall$ and $q_{\mathcal{F}}(x) = \exists$) with respect to $R_{\mathcal{F}}(y) \setminus (D^i_{\mathcal{F}}(x) \cup \{y\})$.*
2. *There exists $z \in D^i_{\mathcal{F}}(x) \setminus \{x\}$ such that $y \in D^{\triangle}_{\mathcal{F}}(z)$.*

Note that this recursive definition is well-founded since $L_{\mathcal{F}}(z) \subset L_{\mathcal{F}}(x)$ for all $z \in D^{\triangle}_{\mathcal{F}}(x) \setminus \{x\}$. In particular, for all $x \in var(\mathcal{F})$, the set $D^{\triangle}_{\mathcal{F}}(x)$ can be computed by successively computing $D^{\triangle}_{\mathcal{F}}(x_1)$, $D^{\triangle}_{\mathcal{F}}(x_2)$, $D^{\triangle}_{\mathcal{F}}(x_3)$, etc., where $\delta_{\mathcal{F}}(x_i) = i$.

For example, recall the QCNF formula $\mathcal{F}$ from above. It holds that $D^{\triangle}_{\mathcal{F}}(u) = \{u\}$, $D^{\triangle}_{\mathcal{F}}(v) = \{u, v\}$, $D^{\triangle}_{\mathcal{F}}(w) = \{u, w\}$, $D^{\triangle}_{\mathcal{F}}(x) = \{u, v, x\}$, $D^{\triangle}_{\mathcal{F}}(y) = \{y\}$, and $D^{\triangle}_{\mathcal{F}}(z) = \{u, w, z\}$. In particular, we obtain $D^{\triangle}_{\mathcal{F}}(z) = \{u, w, z\}$ in the following way: (i) $D^0_{\mathcal{F}}(z) = \{z\}$, (ii) $D^1_{\mathcal{F}}(z) = D^0_{\mathcal{F}}(z) = \{z\}$ since there is no (z, y)-dependency triangle with respect to $\emptyset$ and $D^0_{\mathcal{F}}(z) \setminus \{z\} = \emptyset$, (iii) $D^2_{\mathcal{F}}(z) = D^1_{\mathcal{F}}(z) = \{z\}$ since $D^1_{\mathcal{F}}(z) \setminus \{z\} = \emptyset$, (iv) $D^3_{\mathcal{F}}(z) = D^2_{\mathcal{F}}(z) \cup \{w\} = \{w, z\}$ since there is a (z, w)-dependency triangle with respect to $\emptyset \subseteq \{x, y\}$, (v) $D^4_{\mathcal{F}}(z) = D^3_{\mathcal{F}}(z) = \{w, z\}$ since there is no (z, v)-dependency triangle with respect to $\{x, y\}$ and $v \notin D^{\triangle}_{\mathcal{F}}(w)$ with $D^3_{\mathcal{F}}(z) \setminus \{z\} = \{w\}$, (vi) $D^5_{\mathcal{F}}(z) = D^4_{\mathcal{F}}(z) \cup \{u\} = \{u, w, z\}$ since $u \in D^{\triangle}_{\mathcal{F}}(w)$ with $w \in D^4_{\mathcal{F}}(z) \setminus \{z\}$, and finally (vii) $D^{\triangle}_{\mathcal{F}}(z) = D^5_{\mathcal{F}}(z) = \{u, w, z\}$ since $\delta_{\mathcal{F}}(z) - 1 = 5$.

Theorem 1. *The triangle dependency scheme is indeed a dependency scheme.*

Proof. Let $\mathcal{F}$ be a QCNF formula and $x \in var(\mathcal{F})$. Moreover, let $\mathcal{F}'$ denote $S^{\uparrow}_{D^{\triangle}_{\mathcal{F}}(x)}(\mathcal{F})$. We have to show that $\mathcal{F}$ and $\mathcal{F}'$ are equivalent. To this aim, note that $\mathcal{F}'$ is obtained from $\mathcal{F}$ by quantifier reordering, i.e., by a permutation of the quantifications in the quantifier prefix. It is well known that every permutation of elements can be achieved by successively swapping adjacent elements such that each pair is swapped at most once [12]. In particular, this means that we can transform $\mathcal{F}$ into $\mathcal{F}'$ by successively swapping adjacent quantifications of variables $v \in D^{\triangle}_{\mathcal{F}}(x)$ with variables $w \in L_{\mathcal{F}}(x) \setminus D^{\triangle}_{\mathcal{F}}(x)$, since the relative ordering of variables within these two sets remains unchanged according to the definition of an up-shifting. Thus, it suffices to show that each such elementary transformation step preserves equivalence.

W.l.o.g., let $v \in D^{\triangle}_{\mathcal{F}}(x)$ and $w \in L_{\mathcal{F}}(x) \setminus D^{\triangle}_{\mathcal{F}}(x)$ be two variables with adjacent quantifications in the quantifier prefix that have to be swapped. First note that

$w \notin D^{\triangle}_{\mathcal{F}}(v)$. Otherwise, we obtain $w \in D^{\triangle}_{\mathcal{F}}(x)$ by Definition 4(2), which contradicts our assumption. Thus, we can distinguish between the following cases: If $q_{\mathcal{F}}(v) = q_{\mathcal{F}}(w)$, the equivalence follows trivially. Otherwise, if $q_{\mathcal{F}}(v) \neq q_{\mathcal{F}}(w)$, let us first assume that $q_{\mathcal{F}}(v) = \forall$ and $q_{\mathcal{F}}(w) = \exists$. Let $\mathcal{G}$ denote the formula before v and w are swapped and let $\mathcal{G}'$ denote the formula after v and w have been swapped. In particular, that means $\mathcal{G} = \cdots \exists w \, \forall v \cdots G$ and $\mathcal{G}' = \cdots \forall v \, \exists w \cdots G$. Thus, $\mathcal{G}$ trivially implies $\mathcal{G}'$. For the other direction, we know by Definition 4(1) that there is no (v, w)-dependency triangle with respect to $R_{\mathcal{F}}(w) \setminus (D^{\delta_{\mathcal{F}}(v)-\delta_{\mathcal{F}}(w)-1}_{\mathcal{F}}(v) \cup \{w\}) = R_{\mathcal{G}}(v) \setminus \{v\} = R_{\mathcal{G}'}(w) \setminus \{w\}$. This implies that the set of clauses G can be partitioned into two subsets G_1 and G_2 such that $v \in var(G_1) \setminus var(G_2)$, $\{w, \neg w\} \nsubseteq \bigcup G_1$, and $var(G_1) \cap var(G_2) \subseteq L_{\mathcal{G}'}(w) \setminus \{v\}$. Now assume for the sake of contradiction that the truth value of w depends on the truth value of v when evaluating $\mathcal{G}'$, i.e., there exists a partial truth assignment to the variables in $L_{\mathcal{G}'}(v) \setminus \{v\}$ such that the remaining formula evaluates to true only if w is assigned different truth values for different truth values of v. Let G'_1 and G'_2 be the resulting sets of clauses obtained from G_1 and G_2 respectively after such a partial truth assignment has been applied. Thus, we know that $var(G'_1) \cap var(G'_2) \subseteq \{w\}$. This, however, implies that the clauses in G'_2 must be satisfiable independent of the truth value assigned to w. Moreover, since $\{w, \neg w\} \nsubseteq \bigcup G_1 \supseteq \bigcup G'_1$, we know that the clauses in G'_1 must be satisfiable for a fixed truth value assigned to w, i.e., if $w \in \bigcup G'_1$ then w is assigned 1 and if $\neg w \in \bigcup G'_1$ then w is assigned 0. Thus, the truth value of w can be chosen independently from the truth value of v, which contradicts our assumption. Consequently, swapping v and w in the quantifier prefix of $\mathcal{G}'$ does not affect its truth value. Hence, we know that $\mathcal{G}'$ implies $\mathcal{G}$. The case $q_{\mathcal{F}}(v) = \exists$ and $q_{\mathcal{F}}(w) = \forall$ is completely symmetric. □

Proposition 2. *The triangle dependency scheme is tractable. In particular, given a QCNF formula $\mathcal{F}$ with n variables and length N, we can compute all the sets $D^{\triangle}_{\mathcal{F}}(x)$, with $x \in var(\mathcal{F})$, in time $\mathcal{O}(n^2 N)$.*

Proof. The definition of the triangle dependency scheme gives rise to a recursive algorithm that searches $\mathcal{O}(n^2)$ times for an (x, y)-dependency triangle. The latter search can be accomplished by breadth-first search in time linear in N if appropriate data structures are used. □

Proposition 3. *Let $\mathcal{F}$ be a QCNF formula and $x \in var(\mathcal{F})$. The difference in size of the set $D^{\text{std}}_{\mathcal{F}}(x)$ assigned to x by the standard dependency scheme and the set $D^{\triangle}_{\mathcal{F}}(x)$ assigned to x by the triangle dependency scheme can be arbitrarily large.*

Proof. Let n be an arbitrarily large non-negative integer; w.l.o.g., we can assume that n is odd. Now let $\mathcal{F} = \forall y_1 \, \exists y_2 \, \forall y_3 \ldots \exists y_{n-1} \, \forall y_n \, \exists x \, ((y_1 \vee \neg y_2) \wedge (y_2 \vee \neg y_3) \wedge \cdots \wedge (y_{n-2} \vee \neg y_{n-1}) \wedge (y_{n-1} \vee \neg x) \wedge (x \vee \neg y_n))$. Then $D^{\text{std}}_{\mathcal{F}}(x) = \{x, y_1, \ldots, y_n\}$ and $D^{\triangle}_{\mathcal{F}}(x) = \{x\}$. Hence, $|D^{\text{std}}_{\mathcal{F}}(x)| - |D^{\triangle}_{\mathcal{F}}(x)| = (n + 1) - 1 \geq n$. □

Let us remark that the set assigned to each variable by the triangle dependency scheme may be larger than necessary. Of course, this is not surprising in consideration of Proposition 1. However, we believe that there is a considerable potential for future research to determine dependency schemes which can be computed in a reasonable amount of time and are more general than the triangle dependency scheme.

4 Backdoor Sets

For this section, we consider an arbitrary but fixed dependency scheme D; the definitions of partial assignment trees and backdoor sets are subject to the choice of D.

Partial truth assignments are key for defining backdoor sets of propositional CNF formulas. In the following we introduce the concept of assignment trees which allows us to extend the notions of partial truth assignments and backdoor sets to the quantified setting. We roughly follow a concept of Samulowitz and Bacchus [23].

An *assignment tree* $\mathcal{T} = (T, \lambda)$ is a pair of a rooted binary tree T and a node labeling λ with the following properties. The labeling λ labels every node t (except the root) of T with a pair $\lambda(t) = (x, \varepsilon)$, where x is a variable and $\varepsilon \in \{0, 1\}$. Every node has at most two children. Nodes at the same depth (i.e., distance from the root) are labeled with the same variable and have the same number of children. A variable does not appear at different levels. If a vertex has two children t_1 and t_2, then $\lambda(t_1) = (x, \varepsilon)$ and $\lambda(t_2) = (x, 1 - \varepsilon)$. This completes the definition of an assignment tree.

Let $\mathcal{T} = (T, \lambda)$ be an assignment tree. We denote by $var(\mathcal{T})$ the set of variables occurring in labels of $\mathcal{T}$, and for $x \in var(\mathcal{T})$ we write $\delta_{\mathcal{T}}(x)$ for the depth of x in T. A variable $x \in var(\mathcal{T})$ is *existential* or *universal* in $\mathcal{T}$ if the nodes of T at depth $\delta_{\mathcal{T}}(x) - 1$ have one or two children, respectively. Every leaf t of T corresponds to a truth assignment $\tau : var(\mathcal{T}) \to \{0, 1\}$ consisting of the assignments made along the path from the root to t. We simply write $\tau \in \mathcal{T}$ if τ is such a truth assignment.

Definition 5 (Partial assignment tree). *Let $\mathcal{F}$ be a QCNF formula and $\mathcal{T}$ be an assignment tree. Then $\mathcal{T}$ is a* partial assignment tree of $\mathcal{F}$ *if (i) $var(\mathcal{T}) \subseteq var(\mathcal{F})$ and existential (resp. universal) variables of $\mathcal{T}$ are existentially (resp. universally) quantified variables in $\mathcal{F}$, (ii) $\delta_{\mathcal{T}}(x) < \delta_{\mathcal{T}}(y)$ if and only if $\delta_{\mathcal{F}}(x) < \delta_{\mathcal{F}}(y)$ holds for every pair $x, y \in var(\mathcal{T})$, and (iii) $var(\mathcal{T}) = D_{\mathcal{F}}(var(\mathcal{T}))$.*

We define backdoor sets with respect to some base class $\mathcal{C}$ of QCNF formulas. We think of $\mathcal{C}$ as a class which can be recognized in polynomial time and for which satisfiability can be decided in polynomial time.

Definition 6 (Weak backdoor set). *Let $\mathcal{F}$ be a QCNF formula. The set $B = D_{\mathcal{F}}(X)$ for some $X \subseteq var(\mathcal{F})$ is a* weak backdoor set *of $\mathcal{F}$ with respect to $\mathcal{C}$ (or a* weak $\mathcal{C}$-backdoor set*, for short) if there exists a partial assignment tree $\mathcal{T}$ of $\mathcal{F}$ with $var(\mathcal{T}) = B$ such that $\mathcal{F}[\tau]$ is satisfiable and belongs to $\mathcal{C}$ for all $\tau \in \mathcal{T}$.*

Proposition 4. *Assume that the dependency scheme under consideration is tractable. Let $\mathcal{C} \in \{\text{QHORN}, \text{Q2CNF}\}$ and $k \geq 0$ be a constant. For a given QCNF formula $\mathcal{F}$ we can decide in polynomial time whether $\mathcal{F}$ has a weak $\mathcal{C}$-backdoor set of size at most k. If the answer is affirmative, then $\mathcal{F}$ is satisfiable.*

Proof. (Sketch.) We go through all sets $B \subseteq var(\mathcal{F})$ of size at most k; for $|var(\mathcal{F})| = n$ there are $\mathcal{O}(n^k)$ such sets. For each B we can check in polynomial time whether it is a weak $\mathcal{C}$-backdoor set of $\mathcal{F}$, since the number of partial assignment trees $\mathcal{T}$ with $var(\mathcal{T}) = B$ is a function of k and therefore a constant. □

The running time of the algorithm outlined in the previous proof is polynomial, but the order of the polynomial depends on the size of the backdoor set. Thus, the algorithm is not a fixed-parameter algorithm. We show that it is very unlikely that there exists a fixed-parameter algorithm for this problem. To that end, we consider the following parameterized decision problem with respect to an arbitrary base class $\mathcal{C}$.

WEAK $\mathcal{C}$-BACKDOOR
Instance: A QCNF formula $\mathcal{F}$ and a non-negative integer k.
Parameter: k.
Question: Does $\mathcal{F}$ have a weak $\mathcal{C}$-backdoor set of size at most k?

Proposition 5. *Let* $\mathcal{C} \in \{\text{QHORN}, \text{Q2CNF}\}$. *The problem* WEAK $\mathcal{C}$-BACKDOOR *is* W[2]*-hard.*

Proof. It is easy to see that WEAK $\mathcal{C}$-BACKDOOR for CNF formulas is just a special case of the corresponding problem for QCNF formulas. Hence, the W[2]-hardness result of Nishimura et al. [15] establishes the proposition. □

For the definition of strong backdoor sets, we do not need partial assignment trees as in the case of weak backdoor sets.

Definition 7 (Strong backdoor set). *Let* $\mathcal{F}$ *be a QCNF formula. The set* $B = D_{\mathcal{F}}(X)$ *for some* $X \subseteq var(\mathcal{F})$ *is a* strong backdoor set *of* $\mathcal{F}$ *with respect to* $\mathcal{C}$ *(or a* strong $\mathcal{C}$-backdoor set*, for short) if for all truth assignments* $\tau : B \to \{0, 1\}$ *it holds that* $\mathcal{F}[\tau]$ *belongs to* $\mathcal{C}$.

By taking the size of the backdoor set as the parameter, we obtain the following parameterized decision problem for an arbitrary base class $\mathcal{C}$ of QCNF formulas.

STRONG $\mathcal{C}$-BACKDOOR
Instance: A QCNF formula $\mathcal{F}$ and a non-negative integer k.
Parameter: k.
Question: Does $\mathcal{F}$ have a strong $\mathcal{C}$-backdoor set of size at most k?

For certain base classes it suffices to consider the following variant of backdoor sets.

Definition 8 (Deletion backdoor set). *Let* $\mathcal{F}$ *be a QCNF formula. The set* $B = D_{\mathcal{F}}(X)$ *for some* $X \subseteq var(\mathcal{F})$ *is a* deletion backdoor set *of* $\mathcal{F}$ *with respect to* $\mathcal{C}$ *(or a* deletion $\mathcal{C}$-backdoor set*, for short) if* $\mathcal{F} - B \in \mathcal{C}$.

Note that the various definitions of backdoor sets in this section coincide with their propositional analogons. The problems of detecting (weak or strong) backdoor sets can also be considered as traditional "non-parameterized" problems by taking the parameter as part of the input. These non-parameterized problems are NP-complete, justifying our parameterized approach. Membership follows immediately from Lemma 2 and hardness follows by trivial reduction from the non-quantified propositional versions, which have been shown by Nishimura et al. [15] to be NP-complete.

Lemma 2. *Let $\mathcal{F}$ be a QCNF formula and $\mathcal{C} \in \{\text{QHORN}, \text{Q2CNF}\}$. Then a set $B \subseteq var(\mathcal{F})$ is a strong $\mathcal{C}$-backdoor set of $\mathcal{F}$ if and only if B is a deletion $\mathcal{C}$-backdoor set of $\mathcal{F}$.*

Proof. This result follows again directly from the corresponding result for propositional CNF formulas shown by Nishimura et al. [15]. □

Using a similar construction as for Proposition 3, we can show that the difference between the sizes of the smallest strong backdoor sets using the standard dependency scheme and using the triangle dependency scheme can be arbitrarily large.

5 Detecting Strong Backdoor Sets

In view of Lemma 2, it suffices to develop a fixed-parameter algorithm for detecting deletion backdoor sets with respect to QHORN and Q2CNF; this algorithm is then also a fixed-parameter algorithm for detecting strong backdoor sets.

Our first algorithm searches for a deletion QHORN-backdoor set B of size at most k for a QCNF formula $\mathcal{F}$. For each $x \in var(\mathcal{F})$ we are given the set $D(x) = D_{\mathcal{F}}(x)$. If the dependency scheme D is tractable, then the sets can be computed in polynomial time (see Proposition 2 for the triangle dependency scheme). We can assume that the matrix of $\mathcal{F}$ contains at least one non-Horn clause and that $k \geq 1$, since otherwise the problem has a trivial solution. Consider a non-Horn clause C of $\mathcal{F}$. By definition, C contains at least two positive literals, say x_1 and x_2. Thus, by definition of a deletion backdoor set, we know that either x_1 or x_2 must belong to B. Consequently, we can systematically search for a deletion backdoor set by considering the two cases $x_1 \in B$ and $x_2 \in B$ separately. That is, we search for a deletion backdoor set B_i of size $k_i = k - |D(x_i)|$ for the formula $\mathcal{F}_i = \mathcal{F} - D(x_i)$, $i = 1, 2$. If we find such a backdoor set B_i, then $B = B_i \cup D(x_i)$ is a deletion backdoor set of $\mathcal{F}$. If, however, neither $\mathcal{F}_1$ nor $\mathcal{F}_2$ has such a backdoor set, then $\mathcal{F}$ has no deletion backdoor set of size k. Thus, the problem of finding a deletion backdoor set of size k for $\mathcal{F}$ reduces to two problems of finding deletion backdoor sets of size k_1 for $\mathcal{F}_1$ or of size k_2 for $\mathcal{F}_2$. When we proceed recursively and search for a deletion backdoor set of size at most k' of a formula $\mathcal{F}'$, we must consider variable dependencies with respect to the input formula $\mathcal{F}$ in order to satisfy the closure condition $D_{\mathcal{F}}(B) = B$. Hence we do not need to compute the sets $D_{\mathcal{F}'}(x)$ and can use $D(x) \cap var(\mathcal{F}')$ instead. The pseudo code of the full algorithm **sb-qhorn** is displayed in Figure 1. The algorithm explores a binary search tree of height at most k. Since locating a non-Horn clause and removing variables in $D(x)$ for some $x \in var(\mathcal{F})$ can be done in time linear in the length of the formula, and since the search tree has at most 2^k nodes, we obtain the following result.

Theorem 2. *Given a QCNF formula $\mathcal{F}$ of length N and the sets $D_{\mathcal{F}}(x)$ for $x \in var(\mathcal{F})$. Then we can either find a strong* QHORN*-backdoor set for $\mathcal{F}$ of size at most k or conclude that no such set exists in time $\mathcal{O}(2^k N)$. Consequently, if the dependency scheme under consideration is tractable,* STRONG QHORN-BACKDOOR *is in* FPT.

For the detection of a deletion Q2CNF-backdoor set B of size at most k for a QCNF formula $\mathcal{F}$, we can proceed in a similar fashion. Consider a clause C of $\mathcal{F}$ that contains

Procedure **sb-qhorn**$(\mathcal{F}, k)$
Input: A QCNF formula $\mathcal{F}$ with matrix F, an integer k, and sets $D(x) \subseteq var(\mathcal{F})$ for $x \in var(\mathcal{F})$;
Output: Either a strong QHORN-backdoor set B of size at most k for $\mathcal{F}$, or "no" if such a B does not exist.

1. If $k < 0$, then return "no".
2. If $\mathcal{F} \in$ QHORN, then return $\emptyset$.
3. If $k = 0$, then return "no".
4. Pick a non-Horn clause $C \in F$ and two variables $x_1, x_2 \in var(C) \cap C$.
5. Call **sb-qhorn**$(\mathcal{F} - D(x_1), k - |D(x_1) \cap var(\mathcal{F})|)$.
6. If a set B_1 is returned, then return $B_1 \cup D(x_1)$.
7. Call **sb-qhorn**$(\mathcal{F} - D(x_2), k - |D(x_2) \cap var(\mathcal{F})|)$.
8. If a set B_2 is returned, then return $B_2 \cup D(x_2)$.
9. Return "no".

Fig. 1. Algorithm for detecting strong QHORN-backdoor sets

more than two literals. Let x_1, x_2, x_3 be three different variables occurring in C. By definition of a deletion backdoor set, we know that B must contain at least one x_i, $1 \leq i \leq 3$. Consequently, it suffices to consider three cases, searching for deletion Q2CNF-backdoor sets B_i of size $k_i = k - |D(x_i)|$ for $\mathcal{F}_i = \mathcal{F} - D(x_i)$, $1 \leq i \leq 3$. If such a B_i is found, then $B = B_i \cup D(x_i)$ is a deletion Q2CNF-backdoor set of $\mathcal{F}$. Applying this reasoning recursively yields the algorithm **sb-q2cnf** displayed in Figure 2.

Procedure **sb-q2cnf**$(\mathcal{F}, k)$
Input: A QCNF formula $\mathcal{F}$ with matrix F, an integer k, and sets $D(x) \subseteq var(\mathcal{F})$ for $x \in var(\mathcal{F})$;
Output: Either a strong Q2CNF-backdoor set B of size at most k for $\mathcal{F}$, or "no" if such a B does not exist.

1. If $k < 0$, then return "no".
2. If $\mathcal{F} \in$ Q2CNF, then return $\emptyset$.
3. If $k = 0$, then return "no".
4. Pick a clause $C \in F$ with $|C| \geq 3$ and three variables $x_1, x_2, x_3 \in var(C)$.
5. Call **sb-q2cnf**$(\mathcal{F} - D(x_1), k - |D(x_1) \cap var(\mathcal{F})|)$.
6. If a set B_1 is returned, then return $B_1 \cup D(x_1)$.
7. Call **sb-q2cnf**$(\mathcal{F} - D(x_2), k - |D(x_2) \cap var(\mathcal{F})|)$.
8. If a set B_2 is returned, then return $B_2 \cup D(x_2)$.
9. Call **sb-q2cnf**$(\mathcal{F} - D(x_3), k - |D(x_3) \cap var(\mathcal{F})|)$.
10. If a set B_3 is returned, then return $B_3 \cup D(x_3)$.
11. Return "no".

Fig. 2. Algorithm for detecting strong Q2CNF-backdoor sets

Theorem 3. *Given a QCNF formula $\mathcal{F}$ of length N and the sets $D_{\mathcal{F}}(x)$ for $x \in var(\mathcal{F})$. Then we can either find a strong* Q2CNF*-backdoor set for $\mathcal{F}$ of size at most k or conclude that no such set exists in time $\mathcal{O}(3^k N)$. Consequently, if the dependency scheme under consideration is tractable,* STRONG Q2CNF-BACKDOOR *is in* FPT.

The algorithms outlined above only search for strong backdoor sets but do not decide whether the given QCNF formula $\mathcal{F}$ is true or not. However, if a strong $\mathcal{C}$-backdoor set B for $\mathcal{F}$ of size at most k is found, then we only need to check satisfiability of $\mathcal{F}[\tau] \in \mathcal{C}$ for all $2^{|B|} \leq 2^k$ possible truth assignments $\tau : B \to \{0,1\}$. This follows immediately from the definition of strong backdoor sets. For $\mathcal{C} \in \{\text{QHORN}, \text{Q2CNF}\}$, satisfiability of $\mathcal{F}[\tau] \in \mathcal{C}$ can be decided in polynomial time [10,1].

Theorem 4. *Let $\mathcal{C} \in \{\text{QHORN}, \text{Q2CNF}\}$. The evaluation of QCNF formulas is fixed-parameter tractable with the size of a smallest strong $\mathcal{C}$-backdoor set as parameter.*

6 Conclusion

In this paper we introduced the notion of backdoor sets for quantified Boolean formulas, generalizing the notion from propositional formulas. To this aim, we introduced the notion partial assignment trees, a generalization of partial truth assignments of propositional formulas. An essential part in this paper was devoted to the investigation of dependency schemes which limit the dependency among quantified variables. We proposed a dependency scheme that is both tractable and more powerful than dependency schemes that can be obtained by known methods. We presented fixed-parameter algorithms for detecting strong backdoor sets with respect to quantified Horn and quantified 2CNF formulas. As a consequence, we obtained infinite hierarchies of classes of QCNF formulas that can be recognized and evaluated in uniform polynomial time, with quantified Horn and quantified 2CNF formulas, respectively, at their first level.

References

1. B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.
2. M. Benedetti. Quantifier trees for QBFs. In *Proc. 8th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *LNCS*, pages 378–385. Springer-Verlag, 2005.
3. A. Biere. Resolve and Expand. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, volume 3542 of *LNCS*, pages 59–70. Springer-Verlag, 2005.
4. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
5. U. Egly, H. Tompits, and S. Woltran. On quantifier shifting for quantified Boolean formulas. In *Proc. SAT'02 Workshop on Theory and Applications of Quantified Boolean Formulas*, pages 48–61. Informal Proceedings, 2002.
6. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.
7. J. Hoffmann, C. Gomes, and B. Selman. Structure and problem hardness: Goal asymmetry and DPLL proofs in SAT-based planning. In *Proc. 16th Int. Conf. on Automated Planning and Scheduling (ICAPS'06)*, pages 284–293. AAAI Press, 2006.
8. Y. Interian. Backdoor sets for random 3-SAT. In *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 231–238. Informal Proceedings, 2003.

9. P. Kilby, J. K. Slaney, S. Thiébaux, and T. Walsh. Backbones and backdoors in satisfiability. In *Proc. 20th National Conf. on Artificial Intelligence (AAAI'05)*, pages 1368–1373. AAAI Press, 2005.
10. H. Kleine Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995.
11. H. Kleine Büning and T. Lettman. *Propositional logic: Deduction and algorithms*. Cambridge University Press, 1999.
12. D. E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching, chapter 5.2.2 Sorting by Exchanging, pages 106–110. Addison-Wesley, 1973.
13. I. Lynce and J. P. Marques-Silva. Hidden structure in unsatisfiable random 3-SAT: An empirical study. In *Proc. 16th IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI'04)*, pages 246–251. IEEE Computer Society, 2004.
14. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
15. N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proc. 7th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 96–103. Informal Proceedings, 2004.
16. N. Nishimura, P. Ragde, and S. Szeider. Solving #SAT using vertex covers. In *Proc. 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *LNCS*, pages 396–409. Springer-Verlag, 2006.
17. C. Otwell, A. Remshagen, and K. Truemper. An effective QBF solver for planning problems. In *Proc. Int. Conf. on Modeling, Simulation and Visualization Methods and Int. Conf. on Algorithmic Mathematics and Computer Science (MSV/AMCS'04)*, pages 311–316. CSREA Press, 2004.
18. G. Pan and M. Y. Vardi. Fixed-parameter hierarchies inside PSPACE. In *Proc. 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 27–36. IEEE Computer Society, 2006.
19. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
20. J. Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.
21. Y. Ruan, H. A. Kautz, and E. Horvitz. The backdoor key: A path to understanding problem hardness. In *Proc. 19th National Conf. on Artificial Intelligence (AAAI'04)*, pages 124–130. AAAI Press, 2004.
22. A. Sabharwal, C. Ansotegui, C. Gomes, J. Hart, and B. Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *Proc. 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *LNCS*, pages 382–395. Springer-Verlag, 2006.
23. H. Samulowitz and F. Bacchus. Binary clause reasoning in QBF. In *Proc. 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *LNCS*, pages 353–367. Springer-Verlag, 2006.
24. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. 5th Annual ACM Symposium on Theory of Computing (STOC'73)*, pages 1–9. ACM Press, 1973.
25. S. Szeider. Backdoor sets for DLL subsolvers. *Journal of Automated Reasoning*, 35(1-3): 73–88, 2005.
26. S. Szeider. Generalizations of matched CNF formulas. *Ann. Math. Artif. Intell.*, 43(1-4): 223–238, 2005.
27. R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proc. 18th Int. Joint Conf. on Artificial Intelligence (IJCAI'03)*, pages 1173–1178. Morgan Kaufmann, 2003.
28. R. Williams, C. Gomes, and B. Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Proc. 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 222–230. Informal Proceedings, 2003.

Bounded Universal Expansion for Preprocessing QBF

Uwe Bubeck[1] and Hans Kleine Büning[2]

[1] International Graduate School
Dynamic Intelligent Systems,
Universität Paderborn,
33098 Paderborn, Germany
bubeck@upb.de
[2] Department of Computer Science,
Universität Paderborn,
33098 Paderborn, Germany
kbcsl@upb.de

Abstract. We present a new approach for preprocessing Quantified Boolean Formulas (*QBF*) in conjunctive normal form (*CNF*) by expanding a selection of universally quantified variables with bounded expansion costs. We describe a suitable selection strategy which exploits locality of universals and combines cost estimates with goal orientation by taking into account unit literals which might be obtained.

Furthermore, we investigate how Q-resolution can be integrated into this method. In particular, resolution is applied specifically to reduce the amount of copying necessary for universal expansion.

Experimental results demonstrate that our preprocessing can successfully improve the performance of state-of-the-art *QBF* solvers on well-known problems from the QBFLIB collection.

1 Introduction

Quantified Boolean Formulas (*QBF*) generalize propositional formulas by allowing variables to be quantified either existentially or universally, whereas all variables are implicitly existentially quantified in propositional logic. This enhancement makes *QBF* a concise and natural modeling language for problems in many areas, such as planning, scheduling or verification [13, 15], and many Boolean functions have compact representations in *QBF*.

On the other hand, however, determining the satisfiability of formulas in *QBF* is PSPACE-complete, which is assumed to be significantly harder than the NP-completeness of the propositional SAT problem. But continued research and technical advances have already enabled impressive progress [14] towards the goal of developing powerful *QBF* solvers suitable for practical use. Some of those state-of-the-art solvers (e.g. [10] and [18]) are extensions of the well-known DPLL backtracking search algorithm for propositional logic [8]. Several other techniques have also been successfully applied to *QBF* solving, such as symbolic skolemization [2] or resolution and quantifier expansion [4], to name only a few.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 244–257, 2007.

There are in particular two characteristics of *QBF* which make it so difficult to solve. The first is the fact that for every universal variable, the solver must consider both possible values the variable might have. This obviously affects the DPLL-based search algorithms, but also the other approaches like symbolic skolemization, because an existential variable y_i can be assigned different values depending on the value of a universal whose quantifier precedes the quantifier of y_i. That behavior leads to the second inherent characteristic of *QBF*: the variable ordering imposed by the nesting of the quantifiers must be respected when solving the formula. In fact, the order of the quantifiers matters so much that the complexity of the decision problem for *QBF* formulas in conjunctive normal form (*CNF*) is assumed to become more difficult with each alternation of quantifier blocks in the prefix, resulting in the so-called polynomial hierarchy [12].

QBF instances from various application domains typically have significantly less universal quantifiers than existentials. And if those formulas have multiple alternations of quantifiers, the universal blocks usually tend to be rather short. It therefore appears rewarding to tackle the problem of solving *QBF* formulas by getting rid of the universally quantified variables. After all, a universally quantified formula $\forall x\ \phi(x)$ is just an abbreviation for $\phi(0) \wedge \phi(1)$, where the matrix of the formula is duplicated for x being either 0 or 1. As explained later in more detail, special care has to be taken for existentials which depend on x: those have to be duplicated as well.

This expansion of universal quantifiers has been used successfully for *QBF* solving by Ayari and Basin in QUBOS [1] and in Biere's solver Quantor [4]. Both systems are based on the approach of ultimately expanding all universals and then solving the remaining purely existentially quantified formula with an ordinary SAT solver. In addition, Quantor can also eliminate existential variables by Q-resolution whenever this is cheaper than expansion.

Unfortunately, expanding many universals can quickly lead to rapid growth of the resulting formula. In this paper, we suggest an approach which does not involve eliminating all universals in the formula. Instead, we restrict ourselves to preprocessing *QBF* formulas in *CNF* form by eliminating certain universally quantified variables with bounded expansion costs before feeding the resulting formulas to an ordinary *QBF* solver. The method is based on the idea that we can probably make it significantly easier for the solver when we take out some specially selected cheap or particularly rewarding universals. On the other hand, we avoid the costs of expanding expensive universals which might each require copying almost the whole formula and trigger an exponential explosion. We present a suitable selection strategy which exploits locality of universals and combines cost estimates with goal orientation. Furthermore, we discuss how Q-resolution can be integrated into this method. In particular, we apply resolution specifically to reduce the amount of copying required in a subsequent universal expansion step. This adds another strategic element to our variable elimination procedure. We finish with an experimental evaluation and a conclusion with suggestions for further improvements.

The previous work most closely related to ours is Biere's resolve and expand method [4] as implemented in Quantor. The most obvious difference is that we do only preprocessing with selective expansion under bounded expansion costs. In addition, Quantor only chooses quantifiers from the innermost universal scope for expansion, and we generalize the idea by selecting universals from the whole prefix. To make this work, we use tighter cost estimates and add goal orientation by taking into account unit literals which might be obtained from the expansion. Another major difference is our use of Q-resolution. While Quantor attempts to balance resolution and expansion, we focus specifically on expansion and use resolution only as a strategic means of reducing the expansion costs. Notice that the solver QUBOS which was also mentioned above is very different from both our approach and Quantor. It appears to be geared towards non-*CNF* formulas or circuits and does not perform cost calculations, but just expands the universals in the given order starting with the innermost. Furthermore, QUBOS does not perform Q-resolution at all.

2 Preliminaries

A quantified Boolean formula $\Phi \in QBF$ in *prenex form* is a formula

$$\Phi = Q_1 v_1 ... Q_k v_k \, \phi(v_1, ..., v_k)$$

with quantifiers $Q_i \in \{\forall, \exists\}$ and a propositional formula $\phi(v_1, ..., v_k)$ over variables $v_1, ..., v_k$. We call $Q := Q_1 v_1 ... Q_k v_k$ the *prefix* and ϕ the *matrix* of Φ.

Unless mentioned otherwise, we assume that *QBF* formulas are always in prenex form. In addition, we assume that the matrix is in *conjunctive normal form* (*CNF*), where ϕ is a conjunction of clauses, with each clause being a disjunction of negated or non-negated variables (*literals*).

A universally quantified formula $\forall x \; \phi(x)$ is defined to be true if and only if $\phi(0)$ is true *and* $\phi(1)$ is true. Variables which are bound by universal quantifiers are called *universal variables* and are usually given the names $x_1, ..., x_n$. Similarly, an existentially quantified formula $\exists y \; \phi(y)$ is true iff $\phi(0)$ *or* $\phi(1)$. Variables in the scope of an existential quantifier are *existential variables* and have names $y_1, ..., y_m$. We write $\Phi = Q \; \phi(\mathbf{x}, \mathbf{y})$ or simply $\Phi = Q \; \phi$. Variables which are not bound by quantifiers are *free* variables. In this paper, we do not allow free variables in order to simplify the discussion. But we would like to point out that universal expansion is in fact an equivalence-preserving transformation when formulas with free variables are considered.

Without loss of generality, we require that no variable appears twice in Q (i.e. that all variable names are unique). We call successive quantifiers of the same kind in Q a *quantifier block* S. Blocks are defined to be maximal, such that subsequent blocks S_i and S_{i+1} are always labelled with different kinds of quantifiers. We usually write $\Phi = \forall X_1 \exists Y_1 ... \forall X_r \exists Y_r \; \phi(X_1, ..., X_r, Y_1, ..., Y_r)$ for a *QBF* formula with universal quantifier blocks $\forall X_i = \forall x_{i,1}, ..., x_{i,n_i}$ and existential blocks $\exists Y_i = \exists y_{i,1}, ..., y_{i,m_i}$.

According to their sequence in the prefix, quantifier blocks are ordered linearly $S_1 < ... < S_s$. We call S_s the innermost and S_1 the outermost block. The order of the quantifier blocks also induces a partial order on the variables. Let l_1 and l_2 be two literals in Φ, then we define $l_1 < l_2$ if the variable in l_1 occurs in a quantifier block which precedes the block in which the variable of l_2 appears. If both variables occur in the same block, the order of the literals is undefined.

With $|\Phi|$, we denote the size of a formula $\Phi = Q\ \phi \in QBF$, which we calculate by adding the numbers of literals in all clauses of ϕ. Based on [4], we also introduce notation to describe occurrences of variables and literals in the formula. Given a literal l, we let $o(l)$ denote the number of occurrences of l in a given formula, and $s(l)$ is defined to be the sum of the sizes of all clauses in which l occurs. We further extend the latter notation to sets V of variables by letting $s(V)$ be the sum of the sizes of all clauses in which a variable in V occurs. Consider the example formula $\Phi = \forall x_1 \exists y_1 \exists y_2\ (x_1 \vee \neg y_2) \wedge (\neg y_2 \vee y_1) \wedge \neg y_1$. Here, we have $o(y_1) = 1$, $s(y_1) = 2$ and $o(\neg y_2) = 2$, $s(\neg y_2) = 4$. Furthermore, $s(\{y_1, y_2\}) = 5$.

3 The Basic Preprocessing Algorithm

3.1 Universal Expansion

Consider a QBF formula

$$\Phi = \forall X_1 \exists Y_1 ... \forall X_r \exists Y_r\ \phi(X_1, ..., X_r, Y_1, ..., Y_r)$$

with universal quantifier blocks $\forall X_i = \forall x_{i,1}, ..., x_{i,n_i}$ and existential blocks $\exists Y_i = \exists y_{i,1}, ..., y_{i,m_i}$. In order to expand a universal variable $x_{i,j}$ from the i-th universal block, we have to generate two copies of the matrix ϕ, one where $x_{i,j}$ is 0, and one where $x_{i,j}$ is 1. Furthermore, we must take into account that the existentials in the subsequent blocks $Y_i, ..., Y_r$ depend on $x_{i,j}$ and can have different truth values assigned depending on whether $x_{i,j} = 0$ or $x_{i,j} = 1$. Accordingly, we have to duplicate these existentials to reflect that degree of freedom. We get the expanded formula

$$\begin{aligned}\Phi' = Q'\ &\phi(x_{1,1}, ..., x_{i,j-1}, 0, x_{i,j+1}, ..., x_{r,n_r}, Y_1, ..., Y_r)\ \wedge \\ &\phi(x_{1,1}, ..., x_{i,j-1}, 1, x_{i,j+1}, ..., x_{r,n_r}, Y_1, ..., Y_{i-1}, Y_i', ..., Y_r')\end{aligned}$$

with the new prefix Q' which we obtain from the original prefix when we drop $x_{i,j}$ and replace blocks $\exists Y_k$, $k = i, ..., r$, with $\exists Y_k, Y_k' = \exists y_{k,1}, ..., y_{k,m_k}, y'_{k,1}, ..., y'_{k,m_k}$. Of course, not all clauses are affected by the expansion. An implementation of the algorithm will only have to touch clauses in which $x_{i,j}$ occurs and clauses which must be copied due to the renaming of the existentials.

Strictly adhering to this algorithm might produce lots of redundant copies when universal variables and their dependent existentials are only used locally, which is typical for linearizations of formulas in non-prenex form. Consider the example $\Phi = \forall x_1 \exists y_1 \forall x_2, x_3 \exists y_2, y_3\ \phi(x_1, y_1, x_2, y_2) \wedge \psi(x_1, y_1, x_3, y_3)$. The universal x_1 is used globally in the whole formula, but x_2 and x_3 and the dependent

existentials y_2 and y_3 are only used locally in subformulas ϕ and ψ. However, expanding x_2 with the given procedure would require us to duplicate not only y_2, but also y_3 and clauses in ψ with y_3 in them. Of course, this is redundant, since $\psi(x_1, y_1, x_3, y_3)$ and $\psi(x_1, y_1, x_3, y_3')$ are clearly satisfiability-equivalent.

What we need to do is take into account how variables are actually connected in common clauses. In [4], Biere introduces a suitable concept. His original formulation was not meant for expanding universals from the whole prefix, therefore we have to clarify that universals alone never propagate dependencies (because $\forall v\ (\phi \wedge \psi) \approx (\forall v\ \phi) \wedge (\forall v'\ \psi[v/v'])$). Our formulation is as follows:

We denote a variable v *locally connected* to another variable w if both occur in a common clause, and we write $v \sim w$. Given a universal variable x from the i-th universal quantifier block, we now define

$$\begin{aligned} D_x^{(0)} &:= \{y \in Y_i \cup ... \cup Y_r \mid y \sim x\} \\ D_x^{(k+1)} &:= \{y \in Y_i \cup ... \cup Y_r \mid y \sim y' \text{ for some } y' \in D_x^{(k)}\},\ k \geq 0 \\ D_x &:= \bigcup_k D_x^{(k)} \end{aligned}$$

We call the set D_x the *dependent existentials* of x. When expanding x, we only need to duplicate those existentials and the clauses in which they occur.

3.2 Bounded Expansion

Even when observing locality of universals, repeated application of universal expansion can easily lead to rapid formula growth. It is thus important for our preprocessing to impose strict bounds:

- a *global size limit* C_{global} places an upper bound on the size of the preprocessing output. Variables are only expanded while $|\Phi_{cur}| < C_{global} \cdot |\Phi|$, where Φ is the original input formula (after some initial simplifications as described below) and Φ_{cur} the current formula after some expansions.
  In our experiments, we found rather small values between 2 and 4 to work well without blowing up the formula too much.
- an *individual cost limit* C_{single} is enforced for each single expansion step. We only expand a universal x if the predicted expansion costs c_x (see Section 4) are bounded by $c_x \leq C_{single} \cdot |\Phi_{cur}|$, where Φ_{cur} is the current formula. The idea here is to expand the cheap universals and leave the expensive ones to the solver, since the solver might be able to handle them at lower costs with different strategies.
  We achieved best results with $C_{single} = 0.5$. If no universals have expansion costs below this threshold, the preprocessing will not do anything (except for the initial simplifications). It is due to this strategy of avoiding unfavorable steps that our preprocessing usually does not have noticeable negative effects on the performance of the *QBF* solver.

Listing 1 shows the basic structure of the preprocessor's main loop and illustrates where the bounds are applied. For completeness, we have also included the two occasions where Q-resolution is invoked. This is discussed in Section 5.

Listing 1. The Main Loop of the Preprocessor

```
preprocess (Φ, C_global, C_single) {
  simplify Φ;
  Φ_cur = Φ;
  while (|Φ_cur| < C_global · |Φ|) {
    resolve existentials with negative resolution costs;
    choose universal x with smallest predicted costs c_x;
    if ((x ≠ null) && (c_x ≤ C_single · |Φ_cur|)) {
      reduce dependencies D_x by resolution;
      expand x in Φ_cur;
      simplify Φ_cur;
    } else return Φ_cur;
  }
  return Φ_cur;
}
```

3.3 Simplifications

To reduce the actual costs of universal expansion, we have included the usual simplification rules: unit propagation, pure literal elimination, universal reduction, detection of dual binary clauses and subsumption checking. As they are standard techniques, we do not recall them here and refer the reader to [7, 4].

We apply the rules in a circular fashion where one simplification may trigger the application of another simplification rule, until we reach closure. Initially, we attempt to simplify the whole input formula. Later, we check for specific simplifications as necessary. For the initial simplification, universal reduction is probably the most important operation and allows us to assume for the remaining process that all clauses are cleansed from trailing universal variables which do not dominate any existentials in the same clause. Whenever we later modify clauses or add new ones, we will make sure they are cleansed as well.

In the beginning, we also perform a full subsumption check. Inside the main loop, however, we apply only the cheaper backward subsumption where old clauses are checked for being subsumed by newly generated clauses. The dual case where old clauses might subsume new clauses is not relevant to universal expansion, since expansion never produces longer clauses.

4 Selection Strategy

Making good choices for the universals to be expanded is crucial to the success of the preprocessing. Each expansion of a universal x produces expansion costs $c_x = |\phi'| - |\phi|$, where $|\phi|$ is the size of the matrix of the formula before the expansion and $|\phi'|$ the size of the matrix afterwards, so c_x indicates by how many literals the size of the formula will increase when expanding x. For unsimplified formulas, c_x may also be negative. Since we want to select the universals with the lowest expansion costs, we need a tight cost estimate for each universal in the formula.

4.1 Estimation Scheme

Given a *QBF* formula $\Phi = \forall X_1 \exists Y_1 ... \forall X_r \exists Y_r\ \phi$ with universal quantifier blocks $\forall X_i = \forall x_{i,1}, ..., x_{i,n_i}$ and existential blocks $\exists Y_i = \exists y_{i,1}, ..., y_{i,m_i}$, Quantor [4] estimates the costs of expanding a universal x from the innermost universal quantifier block X_r by considering all existentials in Y_r as dependent on x. Then all clauses in which an existential $y \in Y_r$ occurs need to be duplicated. Using the notation from Section 2, this means that $s(Y_r)$ literals must be added. Furthermore, clauses from the original matrix ϕ in which x occurs negatively are removed from $\phi(x/0)$, as well as clauses in $\phi(x/1)$ where x occurs positively. Finally, all occurrences of x in $\phi(x/0)$ and $\neg x$ in $\phi(x/1)$ are deleted. In total, Quantor's cost estimate is

$$c_x \leq s(Y_r) - s(\neg x) - s(x) - o(x) - o(\neg x)$$

4.2 Including Locality

In our approach, we do not want to restrict ourselves to the innermost universal quantifier block. We also want to be able to expand universals from quantifier blocks X_i with $i < r$. With the cost estimate given above, universals from further outside have higher expansion costs, because we would need to add $s(Y_i \cup ... \cup Y_r)$. Accordingly, choosing universals further outside can only be rewarding if additional factors are considered. For example, it might happen that the expansion of a universal further outside produces valuable unit literals. Or we might encounter the linearization of a non-prenex formula like $\Phi = \forall x_1 \exists y_1\ ((\forall x_2 \exists y_2 \forall x_3 \exists y_3\ \phi)\ \wedge\ (\forall x'_2 \exists y'_2 \forall x'_3 \exists y'_3\ \psi))$. If ϕ and ψ are not balanced in terms of size or difficulty, it may very well make sense to expand, e.g. x_2 before x'_3, although x_2 will be further outside than x'_3 in the linearized prefix.

As described in [4], Quantor's scheduling cannot take into account the locality of universals at this point due to performance considerations. It only uses locality during the actual expansion after a particular universal has already been selected.

Fortunately, our preprocessing scenario requires less frequent scheduling in comparison to a full solver like Quantor. On the one hand, this is due to the fact that we do not have to schedule resolutions. On the other hand, the bounds C_{global} and C_{single} are so tight that we will only expand a rather limited number of universals, therefore we execute much less expansion cycles. Accordingly, we can spend more time on selecting the variables and afford to actually compute the sets D_{x_i} of dependent existentials for the universals x_i in each expansion cycle. This needs time $O(e \cdot m \cdot |\Phi|)$, where e is the number of expansion cycles (iterations of the preprocessor's main loop) and m the number of universals in Φ. Our experiments show that this is still feasible: the total time spent for preprocessing is typically only a small fraction of the time required for the successive run of the solver. Furthermore, we assume that novel data structures like Benedetti's quantifier trees [3] might be applied here with great benefit in future versions of our preprocessor.

Let $D_x \subseteq Y_i \cup ... \cup Y_r$ be the existentials which depend on x. Then we have

$$c_x \leq s(D_x) - s(\neg x) - s(x) - o(x) - o(\neg x)$$

4.3 Goal Orientation

Expanding variables just because it is cheap to do so is a method without much foresight. It turns out that we can further improve our selection strategy by taking into consideration not only costs, but also goals which we might reach by expanding certain universals. A rewarding goal in solving satisfiability problems is to obtain unit literals. Propagating them helps keeping clauses short and might lead to discovering even more unit literals. This is in particular true for formulas with *2-CNF* subformulas, which might just collapse. Consider the following example:

$$\Phi = \forall x_1, x_2 \exists y_1, y_2 \ (x_1 \vee y_1) \wedge (\neg y_1 \vee y_2) \wedge (x_2 \vee \neg y_1 \vee \neg y_2)$$

The universals are pure variables, but we ignore this here for simplicity (perhaps, Φ is embedded into a larger formula). Then $D_{x_1} = D_{x_2} = \{y_1, y_2\}$ and $c_{x_1} = 7 - 2 - 1 = 4$ and $c_{x_2} = 7 - 3 - 1 = 3$, so we expand x_2. After simplifying by removing pure existential literals, we obtain the new matrix $\Phi' = \forall x_1 \exists y_1, y_2 \ (x_1 \vee y_1) \wedge (\neg y_1 \vee y_2) \wedge (\neg y_1 \vee \neg y_2)$ (there are no renamed existentials, because they were simplified away). Had we expanded x_1 instead, the whole matrix would have collapsed to the empty clause after propagating the unit literals y_1 and y_2 when $x = 0$ and removing the pure existentials when $x = 1$. Of course, the same happens when we continue on Φ'. But since our preprocessing only expands a limited number of universals, we might stop after x_2 and miss out on this.

We did not want to have separate measures for expansion costs and benefits, because it would be necessary to balance them somehow. Fortunately, unit literals which are immediately obtained from expanding a universal also have a direct impact on the expansion costs of that universal, as seen in the example. We can therefore simply subtract from the expansion costs the reductions through immediate unit literals, making our cost estimates even tighter and allowing us to continue using costs as our single measure for choosing universals.

In order to do so, we need to know those unit literals. In each iteration of the preprocessor's main loop, we have to perform for each universal variable x a complete unit propagation under the assumption that $x = 0$, and then under the assumption $x = 1$. Since unit propagation can be performed in linear time, this needs $O(e \cdot m \cdot |\Phi|)$, where e is the number of expansion cycles (iterations) and m the number of universals in Φ. As with the calculation of the variable dependencies above, we claim this is still feasible due to the small values of e.

Let U_0 be the unit literals induced by assuming $x = 0$ and U_1 the units when $x = 1$. Then it might happen that U_0 (or analogously U_1) contains unit literals $l_i = \pm y_i$ with existentials y_i whose quantifier precedes the quantifier of x. But since the y_i do not depend on x, such l_i must also be unit literals when $x = 1$.

That means we can propagate those units immediately (and remove them from U_0), even without actually expanding x (similar to [16]). Obtaining units in that way without expansion is a small additional benefit of our unit calculations.

Now let $s_{U_0 \setminus \pm x}$ be the sum of the sizes of all clauses in which a unit literal from U_0 occurs, but not $\pm x$ (we do not want to count those clauses twice). Those clauses are removed from the expansion. Furthermore, let $o_{\neg U_0 \setminus \neg x}$ be the number of clauses in which the negation of a unit literal from U_0 occurs, but not $\neg x$. In those clauses, the negation of the unit literal will be removed. With $s_{U_1 \setminus \pm x}$ and $o_{\neg U_1 \setminus x}$ defined analogously, our cost estimate c_x is finally given as

$$c_x \leq s(D_x) - s_{U_0 \setminus \pm x} - s_{U_1 \setminus \pm x} - o_{\neg U_0 \setminus \neg x} - o_{\neg U_1 \setminus x} - s(\neg x) - s(x) - o(x) - o(\neg x)$$

5 Integrating Q-Resolution

Q-Resolution [11] extends the concept of propositional resolution to *QBF*. We can use it to eliminate an existential variable y in a formula $\Phi \in QBF$ by performing all possible resolutions on y. We can then drop the clauses in which y occurs positively or negatively and replace them with the set of resolvents after performing universal reduction. One problem with this approach is that it may produce large clauses. An even more serious problem is the huge number of resolvents which might be generated when an existential occurs frequently in both phases and we must resolve all positive occurrences with all negative ones.

Accordingly, our preprocessing focuses mainly on universal expansion. Nevertheless, a limited amount of resolution has proven helpful as well. There are two cases when we will apply resolution:

1. Whenever we can eliminate existentials without increasing the formula size.
2. If we can use resolution specifically to reduce costs of a scheduled expansion.

In order to estimate the costs c_y of eliminating an existential y by resolution, we use the upper bound given in [4]:

$$c_y \;\leq\; o(\neg y) \cdot (s(y) - o(y)) \;+\; o(y) \cdot (s(\neg y) - o(\neg y)) \;-\; (s(y) + s(\neg y))$$

At the beginning of each iteration through the preprocessor's main loop, we check whether there are existentials y_i for which this cost estimate c_{y_i} is negative, so that we can be sure not to increase the size of the formula. We then choose the cheapest such existential, i.e. the one for which the cost estimate is the most negative, and eliminate it by resolution. The process is repeated as long as there are existentials with negative cost estimates.

Performing those resolutions before a universal expansion cycle is like a general cleanup that reduces the number of existentials we have to consider and to copy. But we also suggest a more specific application of resolution which only takes place after we have chosen a particular universal x for expansion. Our goal is to reduce its expansion costs c_x. A quick glance at the cost estimates from the last section shows that there are basically two components which determine the value

of c_x: the occurrences of $\pm x$ itself and the occurrences of dependent existentials. We are now going to apply resolution to attack the latter.

The idea is to resolve only on dependent existentials in D_x immediately before expanding x. Eliminating such an $y \in D_x$ yields a double benefit, because we do not only get rid of y itself, but also of its soon-to-be-created copy y'. In addition, we may also save copying some clauses during the following expansion. For example, a clause $(y \vee y_2)$ with $y \in D_x$ and $y_2 \notin D_x$ must be duplicated when x is expanded, but when we resolve on y with $(\neg y \vee y_3)$ and $y_3 \notin D_x$ before expanding x, the resolvent $(y_2 \vee y_3)$ does not need copying, since both literals do not depend on x. Of course, resolution usually produces many resolvents, some of which probably still require copying. In our example, the formula might also contain a clause $(\neg y \vee y_4)$ with $y_4 \in D_x$, so that we obtain a second resolvent $(y_2 \vee y_4)$ which is still dependent on x.

Let δ be an estimate of the average fraction of resolvents which must be duplicated ($0 \leq \delta \leq 1$). Then we can estimate the costs $c_{y|x}$ of resolving an existential $y \in D_x$ before x is expanded:

$$c_{y|x} \;\approx\; (1+\delta) \;\cdot\; (o(\neg y) \cdot (s(y) - o(y)) + o(y) \cdot (s(\neg y) - o(\neg y))) \;-\; 2 \;\cdot\; (s(y) + s(\neg y))$$

We obtain this estimate from the upper bound for resolution given above. The factor 2 reflects the assumption that each clause in which y occurs would have been copied in the subsequent universal expansion (for simplicity, we do not take into account that y and x might occur in common clauses). The factor $(1 + \delta)$ indicates the costs of duplicating in the expansion a portion δ of the resolvents. In our experiments, we found $\delta = 0.5$ to work well when we resolve away all existentials $y \in D_x$ for which the cost estimate $c_{y|x}$ is negative before actually expanding x.

Resolution also reveals an interesting special case. Consider a scenario where we have a universal x_j and two sets D'_{x_j} and D''_{x_j} of existentials which are locally connected to x_j. Further assume that one of those existentials, say $\tilde{y}$, has the property that it constitutes the only link which propagates the local connectivity from x_j and D'_{x_j} on the one hand to D''_{x_j} on the other hand.

Can we destroy that link to make the existentials in D''_{x_j} independent from x_j? If $\tilde{y}$ occurs positively in clauses with existentials from D'_{x_j} and negatively in clauses with existentials from D''_{x_j}, resolving on $\tilde{y}$ will directly link D'_{x_j} and D''_{x_j}, so nothing is gained in this case. But assume $\tilde{y}$ only occurs positively with both D'_{x_j} and D''_{x_j}. Also assume that all negative occurrences of $\tilde{y}$ are in clauses with universals other than x_j and existentials which do not depend on x_j. Now we can resolve away $\tilde{y}$, and the existentials in D'_{x_j} and D''_{x_j} will not be connected anymore, since $\tilde{y}$ has been replaced with variables which do not propagate the dependency.

In this scenario, the special property is that we have an existential y in $D_{x_j} = D'_{x_j} \cup D''_{x_j}$ where one phase of y occurs only in clauses with variables $v \notin D_{x_j}$ and $v \neq x_j$. A closer investigation of this special case reveals that we do not need to perform the actual resolution. Instead, we can simply remove y from D_{x_j}, because it does in fact not depend on x_j: assume that with fixed assignments to $x_1, ..., x_{j-1}$, a given formula Φ is satisfiable if y is assigned different values in

the two cases $x_j = 0$ and $x_j = 1$, i.e. $y = \epsilon$ for $x_j = 0$ and $y = \neg\epsilon$ for $x_j = 1$. Without loss of generality, assume that $\neg y$ is the phase of y which occurs only in clauses with variables $v \notin D_{x_j}$ and $v \neq x_j$. Then none of those clauses contains a variable which depends on x_j, yet those clauses remain satisfied when y flips from ϵ to $\neg\epsilon$ as x_j changes. That means those clauses are true regardless of the value of y. Then we can choose $y = 1$ for both $x_j = 0$ and $x_j = 1$, and all clauses with positive y will be satisfied as well, which means the whole formula is satisfiable. With the obvious argument that if Φ is unsatisfiable when we allow different values for y depending on x_j, this also implies the unsatisfiability of Φ when y must have the same value for $x_j = 0$ and $x_j = 1$, we have the following theorem:

Theorem 1. *Given $\Phi \in QBF$, let x be a universal and y be an existential variable in the scope of x where one phase of y only occurs in clauses with $v \notin D_x$ and $v \neq x$. Then universal expansion of x does not need to duplicate the variable y.*

For performance reasons, our implementation does not check this condition while computing the dependency sets during the scheduling of the expansions, but only prior to executing a scheduled expansion.

6 Implementation and Experiments

We have implemented our preprocessing approach in Java on top of our existing logic framework ProverBox [5, 6]. Using the framework's data structures and basic algorithms has allowed us to quickly build a working preprocessor, although we sacrifice some performance for genericity, as the primary goal of the framework is to integrate different logics and different theorem proving algorithms. In addition, our preprocessor itself it not optimized yet.

The choice of the two bounds C_{global} and C_{single} which control the amount of preprocessing performed (see Section 3.2) obviously has a large impact on the performance of our preprocessor. For space considerations, we do not compare different parameter settings against each other. Instead, we have chosen one successful setting for all of the following experiments.

For the global size limit C_{global} which determines how much larger than the original formula the preprocessed formula may be, we found a value of 2 to perform best. We observed that when formulas are growing, the solver performance is often increasingly dominated by the sheer formula size rather than its complexity. By keeping C_{global} quite low, we try to avoid this effect.

While restricting the resulting formula to about twice the size of the input formula means that in the worst case, only $1 - 2$ universals may be expanded, we can typically expand up to $5 - 10$ of them. For various QBFLIB formulas, the number of expanded universals is even higher: for example, it is usually around 30 for the *ASP* problems, and in some *Adder* formulas, we can expand up to 130 universals. Of course, this number does not include universals merely declared in the prefix, but not used at all in the formula.

The individual cost limit C_{single} determines how expensive a single expansion can be. We achieved best overall results with a value of 0.5, where each expansion is allowed to copy at most half of the current formula. Universals with higher expansion costs are probably better handled by the *QBF* solver itself.

We have conducted our experiments with two state-of-the-art *QBF* solvers, sKizzo [2] and SQBF [17], on an Athlon64 3400+ with 2GB RAM running Windows XP/Cygwin and Java 6. Each solver has been executed under Cygwin in its latest publicly available release and with default parameters. We have applied both solvers, each with and without preprocessing, on a selection of 12 families of benchmarks with a total of 688 instances from the QBFLIB collection [9]. We tried to choose benchmark families of such a difficulty level that the solvers could solve most, but not all formulas of a family within a time limit for each formula of 300 seconds. As expected, when the preprocessor was used, the time limit and the time recorded were for running both the preprocessor and the solver.

As usual, if an instance cannot be solved, e.g. due to a timeout or an out-of-memory condition, it is counted as the timeout value. To make the experiments less time-consuming, we have performed them in a give-up mode where a (sub)family of formulas is quit whenever we encounter an instance solved by neither the solver nor the solver with preprocessor. For example, neither sKizzo itself nor sKizzo with preprocessor can solve the instance *adder-14-sat*, so we skip *adder-16-sat* (without counting it as timeout) and continue with *adder-2-unsat*. Of course, this requires that the instances are approximately sorted in order of ascending difficulty. Where this was not already the case by default, we grouped formulas in obvious subfamilies (e.g. *adder-sat*, *Adder2-sat*, ... or *cnt*, *cnt-r*, ...).

An overview of the results is given in Table 1. It provides for each benchmark family the number of instances solved and the time (in seconds) required by the original solver as well as the combination of preprocessor and solver (sKizzo+pre resp. SQBF+pre). We can observe that both solvers show noticeable overall gains

Table 1. Benchmark Results

		sKizzo		sKizzo+pre		SQBF		SQBF+pre	
Benchmark Family	#inst	solved	time	solved	time	solved	time	solved	time
Adder	32	13	1,568	13	1,655	4	1,203	4	1,201
ASP	40	26	5,181	**40**	1,068	0	12,000	**40**	1,030
Blocks	13	9	368	9	371	10	309	10	323
Connect3 cf_3_3*	21	**7**	381	6	610	11	1,816	**16**	474
Counter	88	52	3,939	**56**	2,563	37	3,101	**38**	2,732
CounterFactual ncf_4*	320	108	1,540	**109**	1,295	87	14,819	**124**	1,244
Evader-Pursuer 4x4-log	7	1	320	1	319	7	13	7	14
k_branch_n	21	**6**	661	5	698	4	455	4	356
k_path_n	21	21	164	21	167	5	1063	**7**	500
RobotsD2 *.2, *.4, *.8	29	10	6,180	**29**	94	20	2,897	**29**	110
Sorting_networks	84	24	1,658	**26**	1,499	9	1,240	**11**	879
Szymanski	12	4	335	4	348	0	300	0	300
Total	688	281	22,295	319	10,687	194	39,216	290	9,163

from the preprocessing: sKizzo+pre could solve 13, 5% more problems in 47, 9% of the time originally recorded, and SQBF+pre did even 49, 5% more problems in 23, 4% of the time. This appears to back our basic assumption that *QBF* solvers can indeed benefit from selectively removing universals beforehand.

For a closer look at the results, we have marked the numbers of solved instances in bold whenever one contestant could solve more problems than the other. We observe that there are only two cases (*k_ branch_ n* and *Connect3* with sKizzo+preprocessing) where the number of solved instances is lower by one with preprocessing. This seems to justify our hypothesis that enforcing tight bounds on the expansion can largely prevent negative effects. On the other hand, there are various families where preprocessing helped solve more problems.

7 Conclusion

Making it easier for *QBF* solvers by selectively removing universal variables in a preprocessing step - a simple yet intriguing idea. We have successfully realized it on the basis of the proven universal expansion method, which we have made bounded and more general by choosing universals from the whole prefix while giving consideration to the locality of variables. In addition, we have added an element of goal orientation to the variable selection by rewarding the generation of unit literals. We have also integrated Q-resolution into our approach and shown how it can be applied specifically to reduce the amount of copying necessary for universal expansion. In concluding experiments with two state-of-the-art solvers on QBFLIB problems, our preprocessing showed noticeable performance gains.

In the future, we would like to evaluate the inclusion of further strategic elements into our variable selection, such as attempting to eliminate complete universal quantifier blocks if they are small, or giving preference to universals that appear in short clauses, or trying to make local areas of the formula completely free of universals. In addition, we will attempt to further optimize our implementation.

References

[1] A. Ayari and D. Basin. *QUBOS: Deciding Quantified Boolean Logic using Propositional Satisfiability Solvers.* Proc. 4th Intl. Conf. on Formal Methods in Computer-Aided Design (FMCAD'02). Springer LNCS 2517, 2002.

[2] M. Benedetti. *Evaluating QBFs via Symbolic Skolemization.* Proc. 11th Intl. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'04). Springer LNCS 3452, 2005.

[3] M. Benedetti. *Quantifier Trees for QBFs.* Proc. 8th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'05). Springer LNCS 3569, 2005.

[4] A. Biere. *Resolve and Expand.* Proc. 7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04). Springer LNCS 3542, 2005.

[5] U. Bubeck. *Design of a Modular Platform for Automated Theorem Proving in Multiple Logics.* M.S. Thesis, San Diego State University, 2003.

[6] U. Bubeck. *ProverBox Automated Reasoning Environment*. Website `http://www.ub-net.de/cms/proverbox.html`, 2006.

[7] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. *An Algorithm to Evaluate Quantified Boolean Formulae and Its Experimental Evaluation*. Journal of Automated Reasoning, 28(2):101–142, 2002.

[8] M. Davis, G. Logemann, and D. Loveland. *A machine program for theorem-proving*. Communications of the ACM, 5(7):394–397, 1962.

[9] E. Giunchiglia, M. Narizzano, and A. Tacchella. *Quantified Boolean Formulas satisfiability library (QBFLIB)*. Website `http://www.qbflib.org`, 2001.

[10] E. Giunchiglia, M. Narizzano, and A. Tacchella. *QUBE: A system for deciding quantified boolean formulas satisfiability*. Proc. 1st Intl. Joint Conf. on Automated Reasoning (IJCAR'01). Springer LNCS 2083, 2001.

[11] H. Kleine Büning, M. Karpinski, and A. Flögel. *Resolution for Quantified Boolean Formulas*. Information and Computation, 117(1):12–18, 1995.

[12] A. Meyer and L. Stockmeyer. *The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space*. Proc. 13th Symp. on Switching and Automata Theory, 1972.

[13] M. Mneimneh and K. Sakallah. *Computing Vertex Eccentricity in Exponentially Large Graphs: QBF Formulation and Solution*. Proc. 6th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'03). Springer LNCS 2919, 2004.

[14] M. Narizzano, L. Pulina, and A. Tacchella. *Report of the Third QBF Solvers Evaluation*. Journal of Satisfiability, Boolean Modeling and Computation, 2:145–164, 2006.

[15] J. Rintanen. *Constructing Conditional Plans by a Theorem-Prover*. Journal of Artificial Intelligence Research, 10:323–352, 1999.

[16] J. Rintanen. *Improvements to the evaluation of quantified Boolean formulae*. Proc. 16th Intl. Joint Conf. on Artificial Intelligence (IJCAI'99). Morgan Kaufmann Publishers, 1999.

[17] H. Samulowitz and F. Bacchus. *Using SAT in QBF*. Proc. 11th Intl. Conf. on Principles and Practice of Constraint Programming. Springer LNCS 3709, 2005.

[18] L. Zhang and S. Malik. *Towards Symmetric Treatment of Conflicts and Satisfaction in Quantified Boolean Satisfiability Solver*. Proc. 8th Intl. Conf. on Principles and Practice of Constraint Programming (CP'02), 2002.

Effective Incorporation of Double Look-Ahead Procedures

Marijn Heule* and Hans van Maaren

Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Sciences
Delft University of Technology
marijn@heule.nl, h.vanmaaren@tudelft.nl

Abstract. We introduce an adaptive algorithm to control the use of the double look-ahead procedure. This procedure sometimes enhances the performance of look-ahead based satisfiability solvers. Current use of this procedure is driven by static heuristics. Experiments show that over a wide variety of instances, different parameter settings result in optimal performance. Moreover, a strategy that yields fast performance on one particular class of instances may cause a significant slowdown on other families. Using a single adaptive strategy, we accomplish performances close to the optimal performances reached by the various static settings. On some families, we clearly outperform even the fastest performance based on static heuristics. This paper provides a description of the algorithm and a comparison with the static strategies. This method is incorporated in march_dl, satz, and kcnfs. Also, the dynamic behavior of the algorithm is illustrated by adaptation plots on various benchmarks.

1 Introduction

Nowadays state-of-the-art satisfiability (SAT) solving shows two main solving architectures: conflict-driven and look-ahead driven. As tuned by the SAT competitions over the last years these two architectures seem to perform in an almost complementary way. The conflict-driven solvers dominate the so called industrial flavored problems (industrial category) while the look-ahead architecture dominates on random problems and problems with an intrinsic combinatorial hardness (part of crafted category). This paper deals with an engineering type of solver optimization with respect to one of the ingredients of look-ahead SAT solving.

The look-ahead architecture of (SAT) solvers has two important features: (1) It selects branching variables that result in a balanced search-tree; and (2) it detects failed literals to reduce the size of the search-tree. Many enhancements have been proposed for this architecture in recent years. One of the enhancements for look-ahead SAT solvers is the DOUBLELOOK procedure, which was

* Supported by the Dutch Organization for Scientific Research (NWO) under grant 617.023.306.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 258–271, 2007.

introduced by Li [7]. The usefulness of this procedure is straight forward: By also performing look-ahead on a second level of propagation, more failed literals could be detected, resulting in an even smaller search-tree.

By always performing additional look-aheads on the reduced formula, the computational costs rise drastically. One would like to restrict this enhancement in such a way that the overall computational time will decrease. Early implementations rely on restrictions based on static heuristics. Although these implementations significantly reduce the time to solve `random 3-`SAT formulas, they yield a clear performance slowdown on many structured instances.

We designed an algorithm for the DOUBLELOOK procedure that adapts towards the (reduced) CNF formula. Our algorithm has some key advantages: 1) Existing DOUBLELOOK implementations require only minor changes; 2) only one magic constant is used, which makes it easy to optimize the algorithm for a specific solver; and 3) this algorithm appears to outperform existing approaches.

In this paper, section 2 provides a general overview of the look-ahead architecture and zooms in on the DOUBLELOOK procedure. Section 3 deals with static heuristics for this procedure and their effect on the performance. Our algorithm is introduced in section 4 together with an alternative by Li. It offers detailed descriptions and motivates the decisions made regarding its design. Section 5 illustrates the usefulness and the behavior of the algorithm by experimental results and adaptation plots. Finally, we draw some conclusions in section 6.

2 Preliminaries

The look-ahead SAT architecture (introduced in [5]) consists of a DPLL search-tree [3] using a LOOKAHEAD procedure to reduce the formula and to determine a branch variable x_{branch} (see algorithm 1). We refer to a look-ahead on literal l as assigning l to true and performing iterative unit propagation. If a conflict occurs during this unit propagation (the empty clause is generated), then l is called a *failed literal* - forcing l to be fixed on false. The resulting formula after a look-ahead on l is denoted by $\mathcal{F}(l = 1)$.

Algorithm 1. DPLL($\mathcal{F}$)

```
 1: if F = ∅ then
 2:    return satisfiable
 3: else if empty clause ∈ F then
 4:    return unsatisfiable
 5: end if
 6: < F; x_branch > := LookAhead( F )
 7: if empty clause ∈ F then
 8:    return unsatisfiable
 9: else if DPLL( F(x_branch = 1) ) = satisfiable then
10:    return satisfiable
11: end if
12: return DPLL( F(x_branch = 0) )
```

The effectiveness of the LookAhead procedure (see algorithm 2) depends heavily on the LookAheadEvaluation function which should favor variables that yield a small and balanced search-tree. Detection of failed literals could further reduce the size of the search-tree. Additionally, several enhancements are developed to boost the performance of Sat solvers based on this architecture.

One of these enhancements is the Preselect procedure, which preselects a subset of the variables (denoted by $\mathcal{P}$) to enter the look-ahead phase. By performing look-ahead only on variables in $\mathcal{P}$ the computational costs of the LookAhead procedure are reduced. However, this may result in less effective branching variables and less detected failed literals. All three solvers discussed in this paper, march_dl, satz, and kcnfs, use a Preselect procedure. Yet, their implementation of this procedure is different.

Another enhancement is the DoubleLook procedure (see algorithm 3), which was introduced by Li [7]. This procedure checks whether a formula resulting from a look-ahead on l is unsatisfiable - it detects l as a failed literal by performing additional look-aheads on the reduced formula. Since the computational costs of these extra unit-propagations are high, this procedure should not be performed on each reduced formula. In the ideal case, one would want to apply it only when the reduced formula could be detected to be unsatisfiable. This requires an indicator expressing the likelihood to observe a conflict.

Let $\mathcal{F}_2$ denote the set of binary clauses of formula $\mathcal{F}$. Li [7] suggests that the number of newly created binary clauses (denoted by $|\mathcal{F}_2 \setminus \mathcal{F}_2^*|$, with $\mathcal{F}_2^*$ referring to the set of binary clauses *before* the reduction) in the reduced formula is an effective indicator whether or not to perform additional look-aheads: If *many* new binary clauses are created during the look-ahead on a literal, the resulting formula is often unsatisfiable. In algorithm 3 the additional look-aheads are triggered when the number of newly created binary clauses exceeds the value of Δ_{trigger}. The optimal value of this parameter is the main topic of this paper.

Algorithm 2. LookAhead($\mathcal{F}$)

```
1: P := Preselect( F )
2: for all variables x_i ∈ P do
3:    F' := DoubleLook( F(x_i = 0), F )
4:    F'' := DoubleLook( F(x_i = 1), F )
5:    if empty clause ∈ F' and empty clause ∈ F'' then
6:       return < F; * >
7:    else if empty clause ∈ F' then
8:       F := F''
9:    else if empty clause ∈ F'' then
10:      F := F'
11:   else
12:      H(x_i) = LookAheadEvaluation( F, F', F'' )
13:   end if
14: end for
15: return < F; x_i with greatest H(x_i) >
```

Algorithm 3. DOUBLELOOK($\mathcal{F}$, $\mathcal{F}^*$)

```
1: if empty clause ∈ F then
2:    return F
3: end if
4: if |F_2 \ F_2^*| > Δ_trigger then
5:    for all variables x_i ∈ P do
6:       F' := F(x_i = 0)
7:       F'' := F(x_i = 1)
8:       if empty clause ∈ F' and empty clause ∈ F'' then
9:          return F'
10:      else if empty clause ∈ F' then
11:         F := F''
12:      else if empty clause ∈ F'' then
13:         F := F'
14:      end if
15:   end for
16: end if
17: return F
```

3 Static Heuristics

The DOUBLELOOK procedure has been implemented in two look-ahead SAT solvers. Initially, Li proposed a static value for Δ_{trigger} [7]: In the first implementation in satz the DOUBLELOOK procedure was triggered using $\Delta_{\text{trigger}} := 65$. (The latest version of satz uses a dynamic algorithm which will be discussed in the next section.) Dubois and Dequen use a variation in their solver kcnfs [4]: In their implementation, the DOUBLELOOK procedure is triggered depending on the original number of variables (denoted by $\#vars$): $\Delta_{\text{trigger}} := 0.18\#vars$.

Both settings of Δ_{trigger} result from optimizing this parameter towards the performance on `random 3-SAT` formulas. On these instances they appear quite effective. However, on structured formulas - industrial and crafted - these settings are far from optimal: On some families, practically none of the look-aheads generate enough new binary clauses to trigger additional look-aheads. Even worse, on many other instances both Δ_{trigger} settings result in a pandemonium of additional look-aheads, which come down hard on the computational costs.

We selected a set of benchmarks from a wide range of families to illustrate these effects. We generated 20 `random 3-SAT` formulas with 350 variables with 1491 clauses (10 satisfiable and 10 unsatisfiable formulas) and used 10 `random 3color` instances from the SAT02 competition [9]. Additionally, we added some crafted and structured instances from various families:

- the `connamacher` family (generic uniquely extendible CSPs) contributed by Connamacher to SAT 2004 [2]. We selected those with $n = 600$ and $d = 0.04$;
- the `ezfact` family (factoring problems) contributed by Pehoushek. We selected the first three benchmarks of 48 bits from SAT 2002 [9];
- the `lksat` family, subfamily `l5k3` (random l-clustered k-SAT instances) contributed by Anton. SAT 2004 [10]. We selected all unsatisfiable instances;

- the `longmult` family (bounded model checking) contributed by Biere [1]. We used the instances of size 8, 10 and 12;
- the `philips` family (multiplier circuit) contributed by Heule to SAT 2004 [10];
- a `pigeon hole` problem (`phole10`) from `www.satlib.org`;
- the `pyhala braun` family (factoring problems) contributed by Pyhala Braun to SAT 2002 [9]. We selected the `unsat-35-4-03` and `unsat-35-4-04`, the two smallest instances from this family not solved during SAT 2002;
- the `stanion/hwb` family (equivalence checking problems) contributed by Stanion. We selected all three benchmarks of size 24 from SAT 2003 [6];
- SAT-encodings of `quasigroup` instances contributed by Zhang [11] We selected the harder unsatisfiable instances - `qg3-9`, `qg5-13`, `qg6-12`, and `qg7-12`.

Besides the random instances, all selected benchmarks are unsatisfiable to realize relatively stable performances. On most these families, the performance of look-ahead SAT solvers is strong[1] (compared to conflict-driven SAT solvers). We performed two tests: One that used constant numbers for Δ_{trigger} - analogue to early satz - and another used values depending on the original number of variables - analogue to kcnfs. For both tests we used the march_dl SAT solver[2]. All experiments were performed on a system with an Intel 3.0 GHz CPU and 1 Gb of memory running on Fedora Core 4. The results of the first test are shown in table 1 and 2, for the low and high values of Δ_{trigger}, respectively.

Recall that satz uses $\Delta_{\text{trigger}} := 65$ - as a result of experiments on `random 3-SAT` instances. As expected, setting $\Delta_{\text{trigger}} := 65$ boosts performances on this family. However, instances from the `pyhala-braun` and `quasigroup` are hard to solve with this parameter setting: On these families the computational time can be reduced by 80% by changing the setting to $\Delta_{\text{trigger}} := 1500$. In general, we observe that a parameter setting which results in optimal performance for a specific family, yields far-from-optimal performances on other families.

Table 3 offers the results of the second test. On `random 3-SAT` optimal performance is realized by $\Delta_{\text{trigger}} := .20\#vars$: Indeed close to the setting used in kcnfs. However, none of the parameter settings result in close-to-optimal performances on all families. Moreover, the optimal performances on the families `3color`, `connamacher`, and `quasigroup` measured during the first test are about twice as fast as the optimal performances of the second test. So, all parameter settings used in the second test are far from optimal - at least for these families.

4 Adaptive DoubleLook

We developed an adaptive algorithm to control the DoubleLook procedure. This algorithm updates Δ_{trigger} after each look-ahead in such fashion, that it adapts towards the characteristics of the (reduced) formula. This section deals with the decisions made regarding the algorithm. First and foremost - for reasons of elegance and practical testing - we focused on using only one magic constant.

[1] based on the results of the SAT competitions, see `http://www.satcompetition.org`
[2] available from `http://www.st.ewi.tudelft.nl/sat/`

Table 1. Performance of march_dl using various static (low) values for Δ_{trigger}

family	0	10	30	65	100	150
3color (10)	118.69	39.91	**31.50**	62.87	67.96	70.42
anton (5)	276.74	269.00	184.73	119.99	80.31	**62.39**
connamacher (3)	5352.55	5407.50	4426.95	4373.89	4559.49	4852.63
ezfact48 (3)	650.01	451.55	287.67	321.70	264.79	187.93
longmult (3)	886.51	578.08	452.34	278.93	**219.35**	255.99
philips (1)	595.43	547.54	391.43	323.97	**273.99**	306.71
pigeon(1)	246.62	**140.05**	141.65	141.45	140.53	140.37
pyhala-braun(2)	4000.0	3024.49	2415.46	2019.37	1481.09	1224.92
quasigroup (4)	2351.98	2102.62	1649.78	1437.39	1362.80	1327.79
stanion (3)	2102.21	1661.80	**941.59**	971.29	964.34	972.18
random-sat (10)	157.12	136.95	96.04	**71.01**	75.44	86.09
random-uns (10)	322.68	285.80	199.03	**143.04**	156.70	178.00

Table 2. Performance of march_dl using various static (high) values for Δ_{trigger}

family	250	400	600	850	1150	1500
3color (10)	67.26	70.26	70.24	70.49	72.21	73.52
anton (5)	64.09	73.28	75.02	75.07	77.22	78.98
connamacher (3)	4353.03	**2633.67**	2642.37	2861.83	4258.05	4099.12
ezfact48 (3)	69.87	**47.54**	55.78	57.16	54.56	51.91
longmult (3)	272.15	291.85	249.99	243.81	278.86	303.99
philips (1)	313.98	317.23	320.84	325.41	328.31	336.90
pigeon(1)	140.61	141.01	140.86	141.38	142.36	142.73
pyhala-braun(2)	1145.64	941.32	607.76	577.75	449.59	**428.26**
quasigroup (4)	1225.14	1011.26	849.64	507.18	455.84	**358.97**
stanion (3)	968.60	963.49	985.46	983.51	988.12	997.59
random-sat (10)	92.53	92.24	93.55	93.20	92.33	91.71
random-uns (10)	186.74	187.64	187.72	189.34	190.04	190.43

The algorithm has three components: (*i*) The Δ_{trigger} initial value, (*ii*) an increment strategy TriggerIncrease and (*iii*) a decrement strategy TriggerDecrease to update Δ_{trigger}. Both strategies consist of two parts: The location within the DoubleLook procedure and the size of the update value.

Regarding the first component: An effective initial value for Δ_{trigger} is probably as hard to determine as an effective global value for this parameter. Therefore, the algorithm should work on many initial values - even on zero, the most costly value at the root node. Hence our decision to initialize $\Delta_{\text{trigger}} := 0$.

The first aspect of the increment strategy is rather straight-forward: Assuming a strong correlation between the value of Δ_{trigger} and the detection of a conflict by the DoubleLook procedure, Δ_{trigger} should always be increased when the procedure fails to meet this objective. Algorithm 4 shows an adaptive variant of the DoubleLook procedure with the increment strategy located at line 17, the first position following a failure.

Table 3. Performance of march_dl using various static values for Δ_{trigger}. These static values are based on the original number of variables (denoted by $\#vars$).

family	.05 $\#vars$	.10 $\#vars$	.15 $\#vars$	.20 $\#vars$	.25 $\#vars$	.30 $\#vars$
3color (10)	**59.08**	67.98	70.19	67.08	68.06	65.87
anton (5)	146.13	83.24	62.67	**59.40**	64.19	67.15
connamacher (3)	4627.01	**4387.70**	4392.15	5078.09	4841.21	4807.81
ezfact48 (3)	324.14	202.17	61.19	50.75	**43.85**	47.64
longmult (3)	**205.46**	247.89	308.71	285.71	265.31	267.09
philips (1)	288.72	**285.43**	311.09	312.46	323.28	311.15
pigeon(1)	158.59	147.60	**142.02**	142.99	143.96	142.15
pyhala-braun(2)	1173.64	1095.74	753.08	590.00	546.79	**484.66**
quasigroup (4)	1473.65	1201.45	1035.91	1069.18	951.36	**837.54**
stanion (3)	1885.25	1110.04	**938.94**	949.83	956.62	973.57
random-sat (10)	118.50	88.57	72.86	**70.61**	71.55	75.97
random-uns (10)	254.60	185.96	155.18	**142.56**	150.69	165.46

The largest reasonable increment of Δ_{trigger} appears to make this parameter equal to the number of newly created binary clauses: Since no conflict was observed, Δ_{trigger} should be at least the number of new binary clauses ($|\mathcal{F}_2 \setminus \mathcal{F}_2^*|$) - which would have prevented the additional computational costs. The smallest value of the increment is a value close to zero and would result in a slow adaptation. The optimal value will probably be somewhere in between. We prefer a radical adaptation. For this reason we use the largest reasonable value:

$$\text{TRIGGERINCREASE}() \ : \ \Delta_{\text{trigger}} := |\mathcal{F}_2 \setminus \mathcal{F}_2^*| \qquad (1)$$

Within the DOUBLELOOK procedure, two events could suggest that Δ_{trigger} should be decreased[3] : (1) The detection of a conflict and (2) the number of newly created binary clauses is less than Δ_{trigger}. The first event seems the most logical: If the DOUBLELOOK procedure detects a conflict, this is a strong indication that a slightly decreased Δ_{trigger} could increase the number of detected failed literals by this procedure. However, this may result in a deadlock situation: The increment strategy could update Δ_{trigger} such that no additional look-ahead will be executed, thereby making it impossible to decrease this parameter.

Placing the decrement strategy after the second event would guarantee that additional look-aheads will be executed every once in a while. Assuming that the computational time could diminish on all benchmarks by the DOUBLELOOK procedure, then this location (algorithm 4 line 19) seems a more appealing choice.

How much should Δ_{trigger} be decreased if after a look-ahead the number of newly created binary clauses is less than this parameter? It seems hard to provide a motivated answer for this question. Therefore, we decided to obtain an effective value for the decrement using experiments.

These experiments were based on two considerations: First, the tests on static heuristics (see section 3) showed that effective parameter settings for Δ_{trigger}

[3] Δ_{trigger} could also be decreased after lines 12 and 14 of algorithm 4: Each new forced literal on a second level of propagation increases the chance of hitting a conflict.

Algorithm 4. AdaptiveDoubleLook($\mathcal{F}$, $\mathcal{F}^*$)

```
 1: if empty clause ∈ F then
 2:    return F
 3: end if
 4: if |F_2 \ F_2^*| > Δ_trigger then
 5:    for all variables x_i ∈ P do
 6:       F' := F(x_i = 0)
 7:       F'' := F(x_i = 1)
 8:       if empty clause ∈ F' and empty clause ∈ F'' then
 9:          TriggerSuccess( )
10:          return F'
11:       else if empty clause ∈ F' then
12:          F := F''
13:       else if empty clause ∈ F'' then
14:          F := F'
15:       end if
16:    end for
17:    TriggerIncrease( )
18: else
19:    TriggerDecrease( )
20: end if
21: return F
```

ranged from 10 to 1500. Therefore, the decrement should not be absolute but relative. So, it should be of the form $\Delta_{\text{trigger}} := c \times \Delta_{\text{trigger}}$ for some $c \in [0, 1]$.

Second, the size of preselected set $\mathcal{P}$ could vary significantly over different nodes. Therefore, the maximum decrement of Δ_{trigger} in each node depends on the size of $\mathcal{P}$. We believe this dependency is not favorable, so we decided to "neutralize" it. Notice that at most $2|\mathcal{P}|$ times in each node Δ_{trigger} could be decreased. Now, let parameter $\text{DL}_{\text{decrease}}$ denote the maximum relative decrement of Δ_{trigger} in a certain node. Then, combining these considerations, the decrement strategy could be formulated as follows:

$$\text{TriggerDecrease}() \ : \ \Delta_{\text{trigger}} := \sqrt[2|\mathcal{P}|]{\text{DL}_{\text{decrease}}} \times \Delta_{\text{trigger}} \tag{2}$$

The "optimal" value for parameter $\text{DL}_{\text{decrease}}$ is discussed in section 5.1.

The latest version of satz (2.15.2) also uses an adaptive algorithm: (*i*) It initializes $\Delta_{\text{trigger}} := .167\#vars$; (*ii*) it increases the Δ_{trigger} using the same TriggerIncrease() placed at the same location. The important difference lies in the location and size of (*iii*) the decreasing strategy: The algorithm realized the decrement at line 9 instead of line 19 of algorithm 4 - so Δ_{trigger} is only reduced after a successful DoubleLook call instead of slowly decrease after each look-ahead.

$$\text{TriggerSuccess}() \ : \ \Delta_{\text{trigger}} := .167\#vars \tag{3}$$

A drawback of this approach is that Δ_{trigger} could never be reduced to a value smaller than $.167\#vars$ - although we noticed from the experiments on static

heuristics that significant smaller values are optimal in some cases (see table 3). When a high value of Δ_{trigger} is optimal this approach might frequently alter between a relative low value ($\Delta_{\text{trigger}} := .167\#vars$) and a relative high value ($\Delta_{\text{trigger}} := |\mathcal{F}_2 \setminus \mathcal{F}_2^*|$) or result in the deadlock situation mentioned above.

5 Results

The adaptive algorithm as described above has been implemented in all look-ahead SAT solvers that contain a DOUBLELOOK procedure: march_dl, satz, and kcnfs. First, we show the effect of parameter $\text{DL}_{\text{decrease}}$ on the computational time. For this purpose, we use the modified march_dl. Second, the performance is compared between the original versions and the modified variants of satz and kcnfs. Third, the behavior of the algorithm is illustrated by adaptation plots. During the experiments we used the benchmarks as described in section 3.

5.1 The Magic Constant

The only undetermined parameter of the adaptive algorithm is $\text{DL}_{\text{decrease}}$. The computational times resulting from various settings for this parameter are shown in table 4. The data shows the effectiveness of the adaptive algorithm:

- Different settings for $\text{DL}_{\text{decrease}}$ result in comparable performances - generally close to the optimal values from the experiments using static heuristics.
- We observe that, for $\text{DL}_{\text{decrease}} := 0.85$, performances are realized for the `anton` and `philips` family that are nearly optimal, while on all the other families this setting outperforms all results using static heuristics.
- The optimal performances achieved by the adaptive heuristics are, on average, about 20% faster than those that are the result of static heuristics.

Table 4. Influence of parameter $\text{DL}_{\text{decrease}}$ on the computational time

family	.75	.80	.85	.90	.95	.99
`3color` (10)	25.77	**25.39**	25.60	28.98	32.79	44.36
`anton` (5)	69.22	67.66	64.99	**63.26**	63.60	66.41
`connamacher` (3)	2258.59	2723.14	**1742.62**	3038.68	2872.84	4431.91
`ezfact48` (3)	39.00	**35.18**	37.87	38.66	38.68	46.08
`longmult` (3)	**197.29**	197.70	203.03	210.12	241.75	258.90
`philips` (1)	307.22	288.10	286.31	**267.17**	280.81	299.71
`pigeon`(1)	**99.31**	99.77	103.47	110.91	113.81	115.28
`pyhala-braun`(2)	369.49	**365.51**	372.98	366.89	376.89	405.05
`quasigroup` (4)	162.38	161.95	157.24	154.59	**150.63**	162.01
`stanion` (3)	**941.94**	946.38	950.44	965.30	984.20	1010.71
`random-sat` (10)	70.04	70.71	**69.21**	69.95	69.74	74.32
`random-uns` (10)	147.40	147.19	**145.95**	148.30	149.17	159.90

Table 5 shows the average values of Δ_{trigger} for various settings of $\text{DL}_{\text{decrease}}$. The average for each family is the mean of the averages of its instances, while for each instance the average is the mean of the averages over all nodes. Because these values are not very accurate, we present only rounded integers.

Parameter $\text{DL}_{\text{decrease}}$ seems to have little impact on these average values. Note that - except for `pyhala-braun` and `quasigroup` instances - the average values of Δ_{trigger} are very close to the optimal values shown in tables 1 and 2. In section 5.3 we provide a possible explanation for the two exceptions.

Table 5. Influence of parameter $\text{DL}_{\text{decrease}}$ on the average value of Δ_{trigger}

family	.75	.80	.85	.90	.95	.99
`3color` (10)	23	24	25	28	33	42
`anton` (5)	129	134	141	162	176	220
`connamacher` (3)	538	575	589	527	462	292
`ezfact48` (3)	324	332	357	370	420	538
`longmult` (3)	76	78	80	90	100	127
`philips` (1)	99	102	107	110	117	142
`pigeon`	7	7	8	8	9	9
`pyhala-braun`(2)	105	108	112	117	127	148
`quasigroup` (4)	537	530	516	489	529	664
`stanion` (3)	21	22	23	25	29	36
`random-sat` (10)	57	58	62	67	77	98
`random-uns` (10)	57	59	62	67	78	98

5.2 Comparison

To test the general application of the adaptive algorithm, we also implemented it in both other SAT solvers that use a DOUBLELOOK procedure: satz and kcnfs. We modified the latest version of the source codes[4]. All three components were made according to the proposed adaptive algorithm: First, initialization is changed to $\Delta_{\text{trigger}} := 0$. Second - only for kcnfs - a line is added to increase Δ_{trigger} when no conflict is detected. Analogue to the march_dl and satz, $\Delta_{\text{trigger}} := |\mathcal{F}_2 \setminus \mathcal{F}_2^*|$.

The third modification is implemented slightly differently, because in satz and kcnfs the size of the pre-selected set $\mathcal{P}$ is computed "on the fly". Therefore, $\sqrt[2|\mathcal{P}|]{\text{DL}_{\text{decrease}}}$ would not be a constant value in each LOOKAHEAD procedure. As a workaround, we decided to use the average value of march_dl for $\sqrt[2|\mathcal{P}|]{\text{DL}_{\text{decrease}}}$ instead. Additionally, from satz the decrement strategy TRIGGERSUCCESS is removed. While using $\text{DL}_{\text{decrease}} := 0.85$, this average appeared approximately 0.9985, which was used for an alternative decrement strategy:

$$\text{TRIGGERDECREASE}() \ : \ \Delta_{\text{trigger}} := 0.9985 \times \Delta_{\text{trigger}} \tag{4}$$

[4] For satz we used version 215.2 (with the adaptive algorithm) which is available at `http://www.laria.u-picardie.fr/~cli/satz215.2.c` and for kcnfs we used the version available at `http://www.laria.u-picardie.fr/~dequen/sat/kcnfs.zip`

Table 6. Comparison between performances of the original and the modified versions of satz, kcnfs and march_dl

	satz		kcnfs		march_dl	
family	original	modified	original	modified	prelim	final
3color (10)	52.71	**36.91**	37.89	**27.88**	72.51	**25.60**
anton (5)	183.97	**123.16**	3433.39	**2382.96**	80.75	**64.99**
connamacher (3)	> 6000	> 6000	4707.51	**4705.23**	4134.85	**1742.62**
ezfact48 (3)	39.96	**32.98**	> 6000	> 6000	54.22	**37.87**
longmult (3)	2411.36	**1582.85**	440.34	**413.19**	265.88	**203.03**
philips (1)	1126.38	**710.75**	750.75	**443.27**	428.52	**286.31**
pigeon(1)	**23.72**	24.12	43.39	**40.25**	145.38	**103.47**
pyhala-braun(2)	1247.46	**881.91**	644.84	**466.92**	380.57	**372.98**
quasigroup (4)	172.40	**171.54**	230.59	**173.86**	351.85	**157.24**
stanion (3)	**3657.49**	3810.53	**3834.31**	3863.13	993.89	**950.44**
random-sat (10)	93.82	**92.56**	**79.63**	80.33	91.63	**69.21**
random-uns (10)	**260.13**	266.81	139.67	**138.22**	189.75	**145.95**

Notice that using value 1.0 instead of 0.9985 would drastically reduce the number of additional look-aheads, because Δ_{trigger} would never be decreased.

The performances of the original and the modified versions of satz, kcnfs, and march_dl are shown in table 6. The proposed adaptive algorithm generally outperforms the one in satz: On most instances from our test, the performance was improved up to 30%, while on the others only small losses were measured. Significant performance boosts are also observed in kcnfs, although the stanion/hwb instances are solved slightly slower. Since we did not optimize the magic constant, additional progress could probably be made.

The double look-ahead is the latest feature of march resulting in version march_dl. The preliminary version used has all features except the DoubleLookAhead procedure. The addition of this feature - using the proposed adaptive algorithm - boost the performance on the complete test set.

5.3 Adaptation Plots

We selected four benchmarks (due to space limitations) to illustrate the behavior of the adaptive algorithm. For each benchmark, the first 10.000 (non-leaf) nodes of the DPLL-tree - using march_dl with $\Delta_{\text{trigger}} := .85$ - are plotted with a colored dot. Nodes are numbered in the (depth-first) order they are visited - so for the first few nodes their number equals their depth. The color is based on the depth of the node in the DPLL-tree. The horizontal axis shows the number of a certain node and the vertical axis shows the average value of parameter Δ_{trigger} in this node. These so-called *adaptation plots* are shown in figures 1, 3, 2 and 4.

In general, we observed that each family has its own kind of adaptation plot, while strong similarities between instances from different families were rare. For none of the tested instances Δ_{trigger} converged to a certain value, which is probably due to the design of the algorithm.

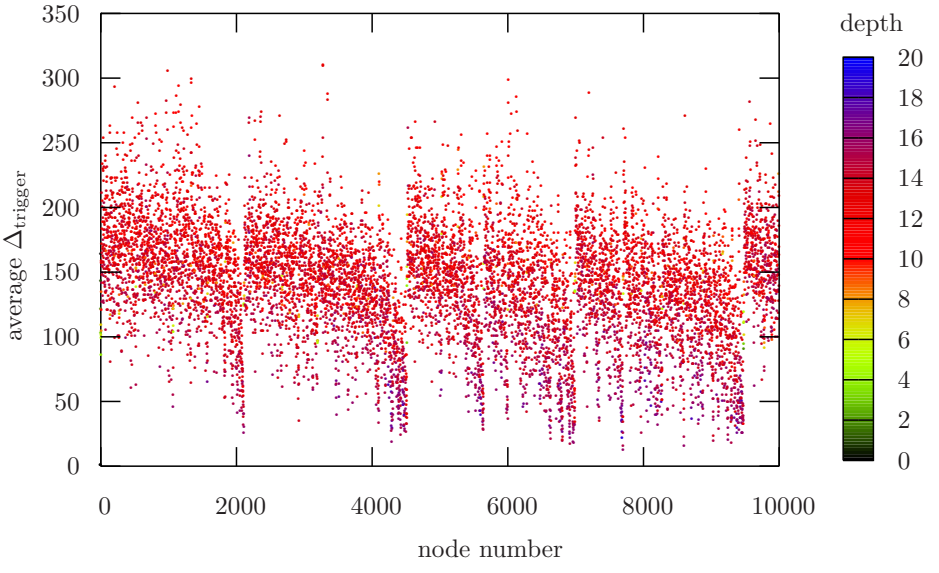


Fig. 1. Adaptation plot of `philips.cnf`

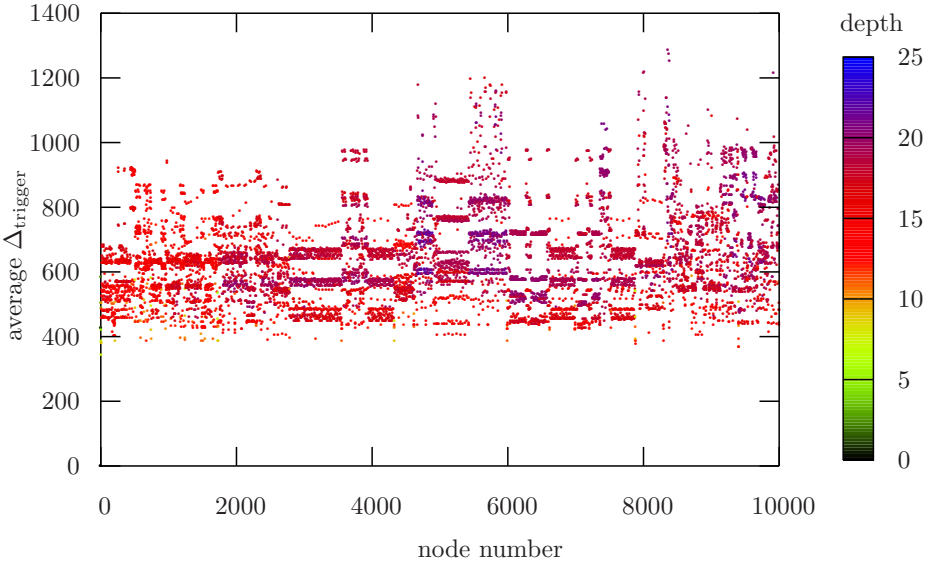


Fig. 2. Adaptation plot of `connm-ue-csp-sat-n600-d0.04-s1211252026.cnf`

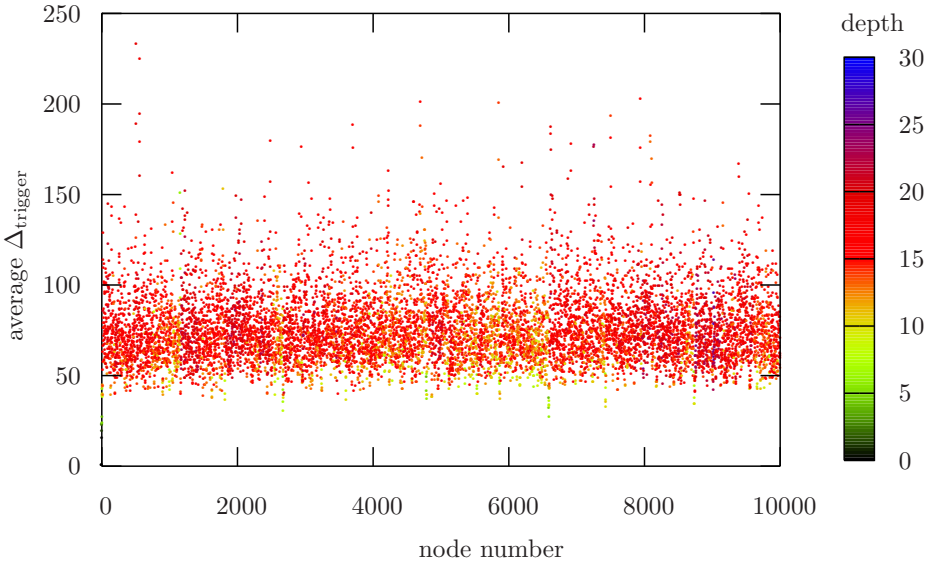


Fig. 3. Adaptation plot of a `random` 3-SAT instance with 350 variables and 1491 clauses

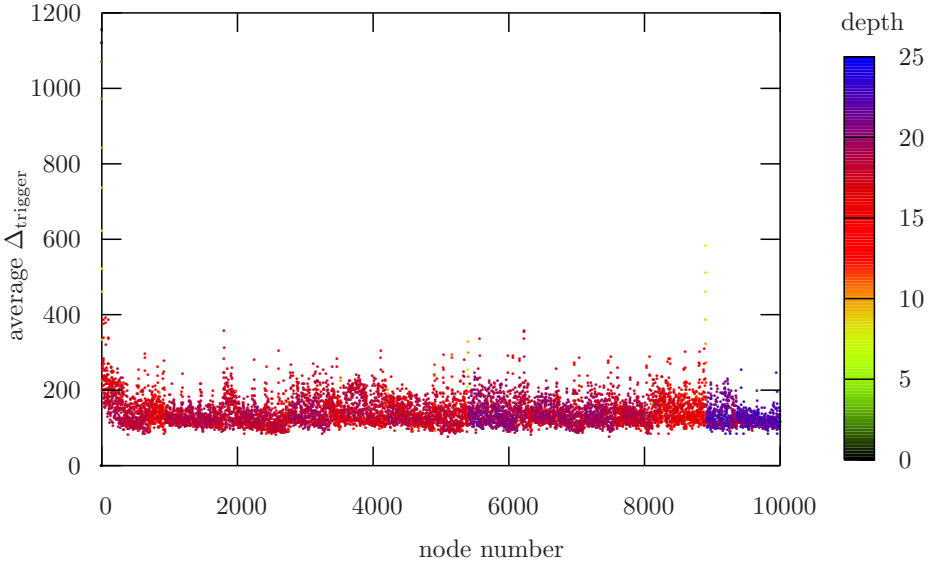


Fig. 4. Adaptation plot of `pyhala-braun-unsat-35-4-03.cnf`

For half of the families, the value of Δ_{trigger} tends to be above average at nodes near the root of the search-tree and / or tends to be below average at nodes near the leafs (see figure 1 and 4). For the other half of the families the opposite trend was noticed (see figure 3 and 2).

Recall that for `pyhala-braun` and `quasigroup` instances the average value for Δ_{trigger} was much lower than the optimum based on static heuristics. Figure 4 offers a possible explanation: Notice that nodes near the root use $\Delta_{\text{trigger}} \approx 1100$ while on average nodes use $\Delta_{\text{trigger}} \approx 100$. Adaptation plots for `quasigroup` instances showed a similar gap. A low static value for Δ_{trigger} will probably result in many additional look-aheads at the nodes near the root which could ruin the overall performance.

6 Conclusions

We presented an adaptive algorithm to control the DoubleLook procedure, which uses - like the static heuristic - only one magic constant. The algorithm has been implemented in all look-ahead Sat solvers that use a DoubleLook procedure. As a result of this modification, all three solvers showed a performance improvement on a wide selection of benchmarks. On macro level we observed that for most instances this algorithm approximates the family specific "optimal" static strategy, while on micro level the algorithm adapts to the (reduced) formula in each node of the search-tree.

References

1. A. Biere, A. Cimatti, E.M. Clarke, Y. Zhu, *Symbolic model checking without BDDs.* in Proc. Int. Conf. TACAS, Springer Verlag, LNCS **1579** (1999), 193–207.
2. H. Connamacher, *A random constraint satisfaction problem that seems hard for DPLL.* In the Proceedings of SAT 2004.
3. M. Davis, G. Logemann, and D. Loveland, *A machine program for theorem proving.* Communications of the ACM **5** (1962), 394–397.
4. O. Dubois and G. Dequen, *source code of the kcnfs solver.* Available at `http://www.laria.u-picardie.fr/~dequen/sat/`.
5. J.W. Freeman, *Improvements to Propositional Satisfiability Search Algorithms.* Ph.D. thesis, University of Pennsylvania, Philadelphia (1995).
6. D. Le Berre and L. Simon, *The essentials of the SAT'03 Competition.* Springer-Verlag, LNCS **2919** (2004), 452–467.
7. C.M. Li, *A constraint-based approach to narrow search trees for satisfiability.* Information processing letters **71** (1999), 75–80.
8. C.M. Li and Anbulagan. *Heuristics Based on Unit Propagation for Satisfiability Problems.* In Proc. of Fifteenth IJCAI (1997), 366–371.
9. L. Simon, D. Le Berre, and E. Hirsch, *The SAT 2002 competition.* Annals of Mathematics and Artificial Intelligence (AMAI) **43** (2005), 343–378.
10. L. Simon, *Sat'04 competition homepage.* `http://www.satcompetition.org/2004`
11. H. Zhang and M.E. Stickel, *Implementing the Davis-Putnam Method.* SAT2000 (2000), 309–326.

Applying Logic Synthesis for Speeding Up SAT

Niklas Een, Alan Mishchenko, and Niklas Sörensson

Cadence Berkeley Labs, Berkeley, USA
EECS Department, University of California, Berkeley, USA
Chalmers University of Technology, Göteborg, Sweden

Abstract. SAT solvers are often challenged with very hard problems that remain unsolved after hours of CPU time. The research community meets the challenge in two ways: (1) by improving the SAT solver technology, for example, perfecting heuristics for variable ordering, and (2) by inventing new ways of constructing simpler SAT problems, either using domain specific information during the translation from the original problem to CNF, or by applying a more universal CNF simplification procedure after the translation. This paper explores preprocessing of circuit-based SAT problems using recent advances in logic synthesis. Two fast logic synthesis techniques are considered: DAG-aware logic minimization and a novel type of structural technology mapping, which reduces the size of the CNF derived from the circuit. These techniques are experimentally compared to CNF-based preprocessing. The conclusion is that the proposed techniques are complementary to CNF-based preprocessing and speedup SAT solving substantially on industrial examples.

1 Introduction

Many of today's real-world applications of SAT stem from formal verification, test-pattern generation, and post-synthesis optimization. In all these cases, the SAT solver is used as a tool for reasoning on boolean circuits. Traditionally, instances of SAT are represented on conjunctive normal form (CNF), but the many practical applications of SAT in the circuit context motivates the specific study of speeding up SAT solving in this setting.

For tougher SAT problems, applying CNF based transformations as a preprocessing step [6] has been shown to effectively improve SAT run-times by (1) minimizing the size of the CNF representation, and (2) removing superfluous variables. A smaller CNF improves the speed of constraint propagation (BCP), and reducing the number of variables tend to benefit the SAT solver's variable heuristic. In the last decade, advances in logic synthesis has produced powerful and highly scalable algorithms that perform similar tasks on circuits. In this paper, two such techniques are applied to SAT.

The first technique, *DAG-aware circuit compression*, was introduced in [2] and extended in [11]. In this work, it is shown that a circuit can be minimized efficiently and effectively by applying a series of local transformations taking logic sharing into account. Minimizing the number of nodes in a circuit tends to reduce the size of the derived CNFs that are passed to the SAT engine. The

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 272–286, 2007.

process is similar to CNF preprocessing where a smaller representation is also achieved through a series of local rewrites.

The second technique applied in this paper is technology mapping for lookup-table (LUT) based FPGAs. Technology mapping is the task of partitioning a circuit graph into cells with k inputs and one output that fits the LUTs of the FPGA hardware, while using as little area as possible. Many of the signals present in the unmapped circuit will be hidden inside the LUTs. In this manner, the procedure can be used to decide for which signals variables should be introduced when deriving a CNF, leading to CNF encodings with even fewer variables and clauses than existing techniques [14,15,9].

The purpose of this paper is to draw attention to the applicability of these two techniques in the context of SAT solving. The paper makes a two-fold contribution: (1) it proposes a novel CNF generation based on technology mapping, and (2) it experimenally demonstrated the practicality of the logic synthesis techniques for speeding up SAT.

2 Preliminaries

A combinational *boolean network* is a directed acyclic graph (DAG) with nodes corresponding to logic gates and directed edges corresponding to wires connecting the gates. Incoming edges of a node are called *fanins* and outgoing edges are called *fanouts*. The *primary inputs* (PIs) of the network are nodes without fanins. The *primary outputs* (POs) are nodes without fanouts. The PIs and POs defines the external connections of the network.

A special case of a boolean network is the *and-inverter graph* (AIG), containing four node types: PIs, POs, two-input AND-nodes, and the constant TRUE modelled as a node with one output and no inputs. Inverters are represented as attributes on the edges, dividing them into *unsigned* edges and *signed* (or complemented) edges. An AIG is said to be *reduced and constant-free* if (1) all the fanouts of the constant TRUE, if any, feeds into POs; and (2) no AND-node has both of its fanins point to the same node. Furthermore, an AIG is said to be *structurally-hashed* if no two AND-nodes have the same two fanin edges including the sign. By decomposing k-input functions into two-input ANDs and inverters, any logic network can be reduced to an AIG implementing the same boolean function of the POs in terms of the PIs.

A *cut* C of node n is a set of nodes of the AIG, called *leaves*, such that any path from a PI to n passes through at least one leaf. A *trivial cut* of a node is the cut composed of the node itself. A cut is *k-feasible* if the number of nodes in it does not exceed k. A cut C is *subsumed* by C' of the same node if $C' \subset C$.

3 Cut Enumeration

Here we review the standard procedure for enumerating all k-feasible cuts of an AIG. Let Δ_1 and Δ_2 be two sets of cuts, and the merge operator $\otimes_k$ be defined as follows:

$$\Delta_1 \otimes_k \Delta_2 \;=\; \{\; C_1 \cup C_2 \;\mid\; C_1 \in \Delta_1,\; C_2 \in \Delta_2,\; |C_1 \cup C_2| \leq k \;\}$$

Further, let n_1, n_2 be the first and second fanin of node n, and let $\Phi(n)$ denote all k-feasible cuts of n, recursively computed as follows:

$$\Phi(n) \;=\; \begin{cases} \Phi(n_1) & ,n \in \text{PO} \\ \{\{n\}\} & ,n \in \text{PI} \\ \{\{n\}\} \;\cup\; \Phi(n_1) \otimes_k \Phi(n_2) & ,n \in \text{AND} \end{cases}$$

This formula gives a simple procedure for computing all k-feasible cuts in a single topological pass from the PIs to the POs. Informally, the cut set of an AND node is the trivial cut plus the pair-wise unions of cuts belonging to the fanins, excluding those cuts whose size exceeds k. Reconvergent paths in the AIG lead to generating subsumed cuts, which may be filtered out for most applications.

In practice, all cuts can be computed for $k \leq 4$. A partial enumeration, when working with larger k, can be achieved by introducing an *order* on the cuts and keeping only the L best cuts at each node. Formally: substitute Φ for Φ_L where $\Phi_L(n)$ is defined as the trivial cut plus the L best cuts of $\Delta_1 \otimes_k \Delta_2$.

4 DAG-Aware Minimization

The concept of DAG-aware minimization was introduced by Bjesse et. al. in [2], and further developed by Mishchenko et. al. in [11]. The method works by making a series of local modifications to the AIG, called rewrites, such that each rewrite reduces the *total* number of AIG nodes. To accurately compute the effect of a rewrite on the total number of nodes, *logic sharing* is taken into account. Two equally-sized implementations of a logical function may have different impact on the total node count if one of them contains a subgraph that is already present in the AIG (see *Figure 1*).

In [11] the authors propose to limit the rewrites to 4-input functions. There exists $2^{16} = 65536$ such functions. By normalizing the order and polarity of input

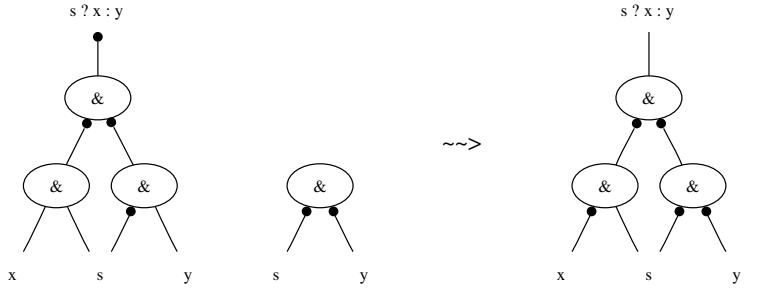


Fig. 1. Given a netlist containing the two fragments on the left, one node can be saved by rewriting the MUX "s ? x : y" to the form on the right, reusing the already present node "$\neg s \wedge \neg y$"

and output variables, these functions are divided into 222 equivalence classes.[1] Good AIG structures, or *candidate implementations*, for these 222 classes can be precomputed and stored in a table. The algorithm of [11] is reviewed below:

> ***DAG-Aware Minimization.*** Perform a 4-feasible cut enumeration, as described in the previous section, proceeding topologically from the PIs to the POs. During the cut enumeration, after computing the cuts for the current node n, try to improve its implementation as follows: For every cut C of n, let f be the function of n in terms of the leaves of C. Consider all the candidate implementations of f and choose the one that reduces the total number of AIG nodes the most. If no reduction is possible, leave the AIG unchanged; otherwise recompute the cuts for the new implementation of node n and continue the topological traversal.

Several components are necessary to implement this procedure:

- A cut enumeration procedure, as described in the previous section.
- A bottom-up topological iterator over the AIG nodes that can handle rewrites during the iteration.
- An incremental procedure for structural hashing. In order to efficiently search for the best substitution candidate, the AIG must be kept structurally-hashed, reduced and constant-free. After a rewrite, these properties may be violated and must be restored efficiently.
- A pre-computed table of good implementations for 4-input functions. We propose to enumerate all structurally-hashed, reduced and constant-free AIGs with 7 nodes or less, discarding candidates not meeting the following property: For each node n, there should be no node m in the subgraph rooted in n, such that replacing n with m leads to the same boolean function. Example: "$(a \wedge b) \wedge (a \wedge c)$" would be discarded since replacing the node "$(a \wedge b)$" with its subnode "b" does not change the function.
- An efficient procedure to evaluate the effect of replacing the current implementation of a node with a candidate implementation.

The implementation of the above components is straight-forward, albeit tedious. We observe that in principle, the topological iterator can be modified to revisit nodes as their fanouts change. When this happens, new opportunities for DAG-aware minimization may be exposed. Modifying the iterator in this way yields an idempotent procedure, meaning that nothing will change if it is run a second time. In practice, we found it hard to make such a procedure efficient.

A simpler and more useful modification to the above procedure is to run it several times with a *perturbation phase* in between. By changing the structure of the AIG, without increasing its size, new cuts can conservatively be introduced with the potential of revealing further node saving rewrites. One way of perturbing the AIG structure is to visit all multi-input conjunctions and modify their decomposition into two-input AND-nodes. Another way is to perform the above minimization algorithm, but allow for zero-gain rewrites.

[1] Often referred to as the NPN-classes, for Negation (of inputs), Permutation (of inputs), Negation (of the output).

5 CNF Through the Tseitin Transformation

Many applications rely on a some version of the Tseitin transformation [14] for producing CNFs from circuits. For completeness, we state the exact version compared against in our experiments. When the transformation is applied to AIGs, two improvements are often used: (1) multi-input ANDs are recognized in the AIG structure and translated into clauses as one gate, and (2) if-then-else expressions (MUXes) are detected in the AIG through simple pattern matching and given a specialized CNF translation. The clauses generated for these two cases are:

$\mathbf{x} \leftrightarrow$ And($\mathbf{a}_1$, $\mathbf{a}_2$, ..., $\mathbf{a}_n$). Clause representation:

$$a_1 \wedge a_2 \wedge \ldots \wedge a_n \rightarrow x$$
$$\overline{a}_1 \rightarrow \overline{x},\ \overline{a}_2 \rightarrow \overline{x},\ \ \ldots\ ,\overline{a}_n \rightarrow \overline{x}$$

$\mathbf{x} \leftrightarrow$ ITE(s,t,f). If-then-else with selector s, true-branch t, false-branch f. Clause representation:

$$s \wedge t \rightarrow x \qquad \overline{s} \wedge f \rightarrow x \qquad \text{(red)}\ \ t \wedge f \rightarrow x$$
$$s \wedge \overline{t} \rightarrow \overline{x} \qquad \overline{s} \wedge \overline{f} \rightarrow \overline{x} \qquad \text{(red)}\ \ \overline{t} \wedge \overline{f} \rightarrow \overline{x}$$

The two clauses labeled "red" are redundant, but including them increases the strength of unit propagation. It should be noted that a two-input XOR is handled as a special case of a MUX with t and f pointing to the same node in opposite polarity. This results in representing each XOR with four three-literal clauses (the redundant clauses are trivially satisfied). In the experiments presented in section 7, the following precise translation was used:

- The *roots* are defined as (1) AND-nodes with multiple fanouts; (2) AND-nodes with a single fanout that is either complemented or leads to a PO; (3) AND-nodes that, together with its two fanin nodes, define an if-then-else.
- If a root node defines an if-then-else, the above translation with 6 clauses, including redundant clauses, is used.
- The remaining root nodes are encoded as multi-input ANDs. The scope of the conjunction rooted at n is computed as follows: Let S be the set of the two fanins of n. While S contains a non-root node, repeatedly replace that node by its two fanins. The above clause translation for multi-input ANDs is then used, unless the conjunction collected in this manner contains both x and $\neg x$, in which case, a unit clause coding for $x \leftrightarrow$ FALSE is used.
- Unlike some other work [7,9], there is no special treatment of nodes that occur only *positively* or *negatively*.

6 CNF Through Technology Mapping

Technology mapping is the process of expressing an AIG in the form representative of an implementation technology, such as standard cells or FPGAs. In

particular, *lookup-table (LUT) mapping* for FPGAs consists in grouping AND-nodes of the AIG into logic nodes with no more than k inputs, each of which can be implemented by a single LUT.

Normally, technology mapping procedures optimize the area of the mapped circuit under delay constraints. Optimal delay mapping can be achieved efficiently [3], but is not desirable for SAT where size matters more than logic depth. Therefore we propose to map for area *only*, in such a way that a small CNF can be derived from the mapped circuit. In the next subsections, we review an improved algorithm for structural technology mapping [12].

6.1 Definitions

A *mapping* $\mathbf{M}$ of an AIG is a partial function that takes a non-PI (i.e. AND or PO) node to a k-feasible non-trivial cut of that node. Nodes for which mapping $\mathbf{M}$ is defined are called *active* (or mapped), the remaining nodes are called *inactive* (or unmapped). A *proper mapping* of an AIG meets the following three criteria: (1) all POs are active, (2) if node n is active, every leaf of cut $\mathbf{M}(n)$ is active, and (3) for every active AND-node m, there is at least one active node n such that m is a leaf of cut $\mathbf{M}(n)$. The *trivial mapping* (or mapping induced by the AIG) is the proper mapping which takes every non-PI node to the cut composed of its immediate fanins.

An *ordered cut-set* Φ_L is a total function that takes a non-PI node to a non-empty ordered sequence of L or less k-feasible cuts. In the next section, $\mathbf{M}$ and Φ_L as will be viewed as updateable objects and treated imperatively with two operations: For an inactive node n, procedure *activate*$(\mathbf{M}, \Phi_L, n)$ sets $\mathbf{M}(n)$ to the first cut in the sequence $\Phi_L(n)$, and then recursively activates inactive leaves of $\mathbf{M}(n)$. Similarly, for an active node n, procedure *inactivate*$(\mathbf{M}, n)$, makes node n inactive, and then recursively inactivates any leaf of the former cut $\mathbf{M}(n)$ that is violating condition (3) of a proper mapping.

Furthermore, *nFanouts*$(\mathbf{M}, n)$ denotes the number of fanouts of n in the subgraph induced by the mapping. The *average fanout* of a cut C is the sum of the number of fanouts of its leaves, divided by the number of leaves. Finally, the *maximally fanout-free cone* (MFFC) of node n, denoted *mffc*$(\mathbf{M}, n)$, is the set of nodes used exclusively by n. More formally, a node m is part of n's MFFC iff every path in the current mapping $\mathbf{M}$ from m to a PO passes through n. For an *inactive* node, *mffc*$(\mathbf{M}, \Phi_L, n)$ is defined as the nodes that would belong to the MFFC of node n if it was first activated.

6.2 A Single Mapping Phase

Technology mapping performs a sequence of refinement phases, each updating the current mapping $\mathbf{M}$ in an attempt to reduce the *total cost*. The cost of a single cut, *cost*(C), is given as a parameter to the refinement procedure. The total cost is defined as sum of *cost*$(\mathbf{M}(n_{act}))$ over all active nodes n_{act}.

Let $\mathbf{M}$ and Φ_L be the proper mapping and the ordered cut-set from the previous phase. A refinement is performed by a bottom-up topological traversal of the AIG, modifying $\mathbf{M}$ and Φ_L for each AND-node n as follows:

- All k-feasible cuts of node n (with fanins n_1 and n_2) are computed, given the sets of cuts for the children: $\Delta = \{\{n\}\} \cup \Phi_L(n_1) \otimes_k \Phi_L(n_2)$
- If the first element of $\Phi_L(n)$ is not in Δ, it is added. This way, the previously best cut is always eligible for selection in the current phase, which is a sufficient condition to ensure global monotonicity for certain cost functions.
- $\Phi_L(n)$ is set to be the L best cuts from Δ, where smaller cost, higher average fanout, and smaller cut size is better. The best element is put first.
- If n is active in the current mapping $\mathbf{M}$, and if the first cut of $\Phi_L(n)$ has changed, the mapping is updated to reflect the change by calling *inactivate*($\mathbf{M}$, n) followed by calling *activate*($\mathbf{M}$, Φ_L, n). After this, $\mathbf{M}$ is guaranteed to be a proper mapping.

6.3 The Cost of Cuts

This subsection defines two complementary heuristic cost function for cuts:

Area Flow. This heuristic estimates the *global* cost of selecting a cut C by recursively approximating the cost of other cuts that have to be introduced in order to accommodate cut C:

$$cost_{AF}(C) = area(C) + \sum_{n \in C} \frac{cost_{AF}(first(\Phi_L(n)))}{\max(1, nFanouts(\mathbf{M}, n))}$$

Exact Local Area. For nodes currently not mapped, this heuristic computes the total cost-increase incurred by activating n with cut C. For mapped nodes, the computations is the same but n is first deactivated. Formally:

$$mffc(C) = \bigcup_{n \in C} mffc(\mathbf{M}, \Phi_L, n)$$

$$cost_{ELA}(C) = \sum_{n \in mffc(C)} area(first(\Phi_L(n))$$

In standard FPGA mapping, each cut is given an area of 1 because it takes one LUT to represent it. A small but important adjustment for CNF generation is to define area in terms of the number of *clauses* introduced by that cut. Doing so affects both the area flow and the exact local area heuristic, making them prefer cuts with a small representation.

The boolean function of a cut is translated into clauses by deriving its *irredundant sum-of-products (ISOP)* using Minato-Morreale's algorithm [10] (reviewed in *Figure 2*). ISOPs are computed for both f and $\neg f$ to generate clauses for both sides of the bi-implication $t \leftrightarrow f(x_1, \ldots, x_k)$. For the sizes of k used in the experiments, boolean functions are efficiently represented using truth-tables. In practice, it is useful to impose a bound on the number of products generated and abort the procedure if it is exceeded, giving the cut an infinitly high cost.

```
cover isop(boolfunc L, boolfunc U)
{
    if (L == FALSE) return ∅
    if (U == TRUE) return {∅}

    x = topVariable(L, U)
    (L0, L1) = cofactors(L, x)
    (U0, U1) = cofactors(U, x)

    c0    = isop(L0 ∧ ¬U1, U0)
    c1    = isop(L1 ∧ ¬U0, U1)
    Lnew  = (L0 ∧ ¬func(c0)) ∨ (L1 ∧ ¬func(c1))
    c*    = isop(Lnew, U0 ∧ U1)

    return ({x} × c0) ∪ ({¬x} × c1) ∪ c*
}
```

Fig. 2. *Irredundant sum-of-product generation.* A ***cover*** (= SOP = DNF) is a set, representing a disjunction, of *cubes* (= product = conjunction of literals). A cover c induces a boolean function $func(c)$. An *irredundant SOP* is a cover c where no cube can be removed without changing $func(c)$. In the code, ***boolfunc*** denotes a boolean function of a fixed number of variables $x_1, x_2, \ldots, x_n$ (in our case, the width of a LUT). L and U denotes the lower and upper bound on the cover to be returned. At top-level, the procedure is called with $L = U$. Furthermore, *topVariable(L, U)* selects the first variable, from a fixed variable order, which L or U depends on. Finally, *cofactors*(F, x) returns the pair $(F[x = 0], F[x = 1])$.

6.4 The Complete Mapping Procedure

Depending on the time budget, technology mapping may involve different number of refinement passes. For SAT, only a very few passes seem to pay off. In our experiments, the following two passes were used, starting from the trivial mapping induced by the AIG:

- An initial pass, using the area-flow heuristic, $cost_{AF}$, which captures the global characteristics of the AIG.
- A final pass with the exact local area heuristic, $cost_{ELA}$. From the definition of local area, this pass cannot increase the total cost of the mapping.

Finally, there is a trade-off between the quality of the result and the speed of the mapper, controlled by the cut size k and the maximum number of cuts stored at each node L. To limit the scope of the experimental evaluation, these parameters were fixed to $k = 8$ and $L = 5$ for all benchmarks. From a limited testing, these values seemed to be a good trade-off. It is likely that better results could be achieved by setting the parameters in a problem-dependent fashion.

7 Experimental Results

To measure the effect of the proposed CNF reduction methods, 30 hard SAT problems represented as AIGs were collected from three different sources. The

first suite, "Cadence BMC", consists of internal Cadence verification problems, each of which took more than one minute to solve using SMV's BMC engine. Each of the selected problem contains a bug and has been unrolled upto the length k, which reveals this bug (yielding a satisfiable instance) as well as upto length $k-1$ (yielding an unsatisfiable instance).

The second suite, "IBM BMC", is created from publically available IBM BMC problems [16]. Again, problems containing a bug were selected and unrolled to length k and $k-1$. Problems that MiniSat could not solve in 60 minutes were removed, as were problems solved in under 5 seconds.

Finally, the third suite, "SAT Race", was derived from problems of *SAT-Race 2006*. Armin Biere's tool "cnf2aig", part of the AIGER package [1], was applied to convert the CNFs to AIGs. Among the problems that could be completely converted to AIGs, the "manol-pipe" class were the richest source. As before, very hard and very easy problems were not considered.

For the experiments, we used the publically available synthesis and verification tool ABC [8] and the SAT solver MiniSat2. The exact version of ABC used in these experiments, as well as other information useful for reproducing the experimental results presented in this paper, can be found at [5].

Clause Reduction. In *Table 1* we compare the difference between generating CNFs using only the Tseitin encoding (section 5) and generating CNFs by applying different combinations of the presented techniques, as well as CNF preprocessing [6] (as implemented in MiniSat2). Reductions are measured against the Tseitin encoding. For example, a reduction of 62% means that, on average, the transformed problem contains 0.38 times the original number of clauses.

We see a consistent reduction in the CNF size, especially in the case where the CNF was derived using technology mapping. The preprocessing scales well, although its runtime, in our current implementation, is not negligible.

For space reasons, we do not present the total number of literals. However, we note that: (1) the speed of BCP depends on the number of clauses, not literals; (2) deriving CNFs from technology mapping produces clauses of at most size $k+1$, which is 9 literals in our case; and (3) in [6] it was shown that CNF preprocessing in general does not increase the number of literals significantly.

SAT Runtime. In *Table 2* we compare the SAT runtimes of the differently preprocessed problems. Runtimes do *not* include preprocessing times. At this stage, when the preprocessing has not been fully optimized for the SAT context, it is arguably more interesting to see the potential speedup. If the preprocessing is too slow, its application can be controlled by modifying one of the parameters (such as the number or width of cuts computed), or preprocessing may be delayed until plain SAT solving has been tried for some time without solving the problem. Furthermore, for BMC problems, the techniques can be applied before unrolling the circuit, which is significantly faster (see *Incremental BMC* below).

Speedup is given both as a total speedup (the sum total of all SAT runtimes) and as arithmetic and harmonic average of the individual speedups. For BMC, we see a clear gain in the proposed methods, most notably for the Cadence

BMC problems where a total speedup of 6.9x was achieved not using SATELITE-style preprocessing, and 5.3x with SATELITE-style preprocessing (for a total of 22.3x speedup compared to plain SAT on Tseitin). However, the problems from the SAT-Race benchmark exhibit a different behavior resulting in an increased runtime. It is hard to explain this behavior without knowing the details of the benchmarks. For example, equivalence checking problems are easier to solve if the equivalent points in the modified and golden circuit are kept. The proposed methods may remove such pairs, making the problems harder for the SAT solver.

CNF Generation based on Technology Mapping. Here we measure the effect of using the number of CNF clauses as the size estimator of a LUT, rather than a unit area as in standard technology mapping. In both cases, we map using LUTs of size 8, keeping the 5 best cuts at each node during cut enumeration. The results are presented in *Table 5*. As expected, the proposed technique lead to fewer *clauses* but more *variables*. In these experiments, the clause reduction almost consistently resulted in shorter runtimes of the SAT solver.

Incremental BMC. An alternative and cheaper use of the proposed techniques in the context of BMC, is to minimize the AIG before unrolling. This prevents simplification across different time frames, but is much faster (in our benchmarks, the runtime was negligible). The clause reduction and the SAT runtime using DAG-aware minimization are given in *Table 4*. In this particular experiment, ABC was not used, but an in-house Cadence implementation of DAG-aware minimization and incremental BMC. Ideally, we would like to test the CNF generation based on technology mapping as well, but this is currently not available in the Cadence tool. For licence reasons, IBM benchmarks could not be used in this experiment. Instead, 5 problems from the TIP-suite [1] were used, but they suffer from being too easy to solve.

8 Conclusions

The paper explores logic synthesis as a way to speedup the solving of circuit-based SAT problems. Two logic synthesis techniques are considered and experimentally evaluated. The first technique applies recent work on DAG-aware circuit compression to preprocess a circuit before converting it to CNF. In spirit, the approach is similar to [4]. The second technique directly produces a compact CNF through a novel adaptation of area-oriented technology mapping, measuring area in terms of CNF clauses.

Experimental results on several sets of benchmarks have shown that the proposed techniques tend to substantially reduce the runtime of SAT solving. The net result of applying both techniques is a 5x speedup in solving for hard industrial problems. At the same time, some slow-downs were observed on benchmarks from the previous year's SAT Race. This indicates that more work is needed for understanding the interaction between the circuit structure and the heuristics of a modern SAT-solver.

Acknowledgements

The authors acknowledge helpful discussions with Satrajit Chatterjee on technology mapping and, in particular, his suggestion to use the average number of fanins' fanouts as a tie-breaking heuristic in sorting cuts.

Table 1. *CNF generation with different preprocessing.* "**(orig)**" denotes the original Tseitin encoding; "**D**" DAG-Aware minimization; "**T**" CNF generation through Technology Mapping; "**S**" SatELite style CNF preprocessing. On the left, the number of clauses in the CNF formulation is given, in thousands. On the right, the runtimes of applied preprocessing are summed up. No column for the time of generating CNFs through Tseitin encoding is given, as they are all less than a second. The "Cdn" problems are internal Cadence BMC problems; the "ibm" problems are IBM BMC problems from [16]; the remaining ten problems are the "manol-pipe" problems from SAT-Race 2006 [13] back-converted by "cnf2aig" into the AIG form.

	Clause Reduction (k clauses)								**Preprocessing Time** (sec)						
Problem	(orig)	S	D	DS	T	TS	DT	DTS	S	D	DS	T	TS	DT	DTS
Cdn1-70u	160	113	69	43	54	41	36	29	1	6	7	14	15	11	12
Cdn1-71s	166	117	71	44	55	43	37	30	1	6	6	14	15	12	12
Cdn2-154u	682	452	467	310	312	257	282	254	6	31	35	48	51	66	68
Cdn2-155s	693	459	475	316	318	262	287	259	7	32	36	49	52	67	69
Cdn3.1-18u	1563	813	952	511	905	529	506	306	12	91	99	151	159	189	193
Cdn3.1-19s	1686	898	1028	559	977	593	547	336	12	98	107	162	170	208	212
Cdn3.2-19u	1684	899	1027	561	977	578	547	337	12	98	106	163	171	206	210
Cdn3.2-20s	1807	979	1102	611	1049	612	588	368	13	104	114	175	184	219	224
Cdn3.3-19u	1686	897	1027	560	977	578	547	338	12	100	109	163	171	204	208
Cdn3.3-20s	1809	974	1103	611	1049	647	588	368	14	104	113	174	183	224	229
ibm18-28u	151	95	72	55	67	54	50	48	1	5	6	11	11	11	12
ibm18-29s	158	99	75	57	70	56	53	50	1	5	6	11	12	12	12
ibm20-43u	253	156	127	97	120	99	89	85	2	10	11	19	20	20	21
ibm20-44s	259	161	131	100	123	101	91	88	2	10	11	19	20	21	21
ibm22-51u	415	269	211	160	201	174	149	143	4	16	17	31	33	33	34
ibm22-52s	425	275	216	164	205	178	153	147	4	16	18	32	33	34	34
ibm23-35u	231	147	116	86	100	85	80	76	2	9	9	17	18	18	19
ibm23-36s	239	152	120	89	103	89	83	78	2	9	10	17	18	19	19
ibm29-25u	53	35	28	21	22	20	18	17	0	2	2	4	4	5	5
ibm29-26s	55	36	29	22	24	21	19	18	0	2	3	5	5	5	5
c10id-s	293	273	280	258	177	159	167	151	2	20	21	31	33	46	48
c10nidw-s	643	593	612	563	416	380	394	363	4	47	52	77	84	119	126
c6nidw-i	154	142	147	134	97	89	93	87	1	10	11	18	19	26	27
c7b	41	36	39	33	27	26	26	25	0	3	3	5	6	7	8
c7b-i	40	36	38	33	27	26	26	25	0	3	4	5	5	7	8
c9	23	20	20	17	15	14	13	12	0	2	2	3	3	4	4
c9nidw-s	535	489	507	465	340	312	326	300	4	39	42	66	71	96	101
g10b	128	116	127	111	87	82	83	76	1	9	10	15	16	23	24
g10id	258	240	254	234	161	147	156	143	2	20	21	30	32	47	49
g7nidw	119	110	118	107	78	72	75	70	1	8	8	13	14	20	21
Avg. red.	–	29%	32%	47%	46%	56%	57%	62%							

Table 2. *SAT runtime with different preprocessing.* "**(orig)**" denotes the original Tseitin encoding; "**D**" DAG-Aware minimization; "**T**" CNF generation through Technology Mapping; "**S**" SATELITE style CNF preprocessing. Given times do *not* include preprocessing, only SAT runtimes. Speedups are relative to the "**(orig)**" column.

Problem	**SAT Runtime** (sec) – Cadence BMC (orig)	S	D	DS	T	TS	DT	DTS
Cdn1-70u	21.9	12.3	3.6	3.1	2.5	4.1	**1.2**	1.3
Cdn1-71s	15.2	8.8	7.7	3.9	**2.1**	3.1	4.0	2.7
Cdn2-154u	116.4	48.3	41.1	37.7	11.6	34.4	15.6	**9.3**
Cdn2-155s	101.8	22.9	12.9	16.2	18.2	50.6	13.4	**6.9**
Cdn3.1-18u	1516.0	139.4	361.9	119.4	196.3	78.8	78.8	**39.0**
Cdn3.1-19s	1788.2	276.7	535.0	154.8	317.8	137.1	131.9	**42.5**
Cdn3.2-19u	403.8	214.4	239.8	169.7	140.9	**73.7**	114.8	78.1
Cdn3.2-20s	3066.1	893.4	1002.9	353.2	376.2	313.5	687.5	**96.5**
Cdn3.3-19u	316.1	225.6	133.9	104.7	107.9	107.6	**53.2**	55.0
Cdn3.3-20s	2305.4	456.4	863.1	385.8	507.0	236.9	307.2	**101.2**
Total speedup:		4.2x	3.0x	7.2x	5.7x	9.3x	6.9x	22.3x
Arithmetic average speedup:		3.9x	3.6x	6.5x	6.3x	7.6x	9.2x	19.7x
Harmonic average speedup:		2.7x	2.9x	4.8x	5.3x	4.9x	6.6x	11.5x

Problem	**SAT Runtime** (sec) – IBM BMC (orig)	S	D	DS	T	TS	DT	DTS
ibm18-28u	83.7	82.6	39.2	41.9	45.0	54.2	23.2	**18.5**
ibm18-29s	93.6	47.6	46.8	25.1	36.9	23.5	25.9	**20.9**
ibm20-43u	805.5	890.1	402.3	488.0	540.3	283.6	219.9	**215.1**
ibm20-44s	1260.2	278.4	305.6	**83.8**	277.2	422.2	265.7	303.6
ibm22-51u	361.8	194.6	109.2	88.6	145.8	170.8	**67.0**	82.5
ibm22-52s	408.4	489.0	148.3	135.7	187.2	177.9	120.5	**91.3**
ibm23-35u	540.3	365.9	264.2	241.5	260.1	220.2	181.4	**130.7**
ibm23-36s	856.2	743.4	527.9	356.8	436.2	585.7	**144.7**	238.1
ibm29-25u	329.7	375.6	39.0	29.4	42.9	56.6	28.5	**11.4**
ibm29-26s	71.3	190.5	41.7	**20.9**	71.5	31.5	28.0	25.4
Total speedup:		1.3x	2.5x	3.2x	2.4x	2.4x	4.4x	4.2x
Arithmetic average speedup:		1.5x	3.0x	4.9x	2.8x	2.8x	4.7x	6.5x
Harmonic average speedup:		1.0x	2.4x	3.1x	2.1x	2.4x	4.0x	4.3x

Table 3. *SAT runtime with different preprocessing (cont. from Table 2)*

Problem	**SAT Runtime** (sec) – SAT Race (orig)	S	D	DS	T	TS	DT	DTS
c10id-s	26.7	**5.1**	25.1	23.6	50.6	25.2	49.8	14.7
c10nidw-s	710.5	624.7	700.3	880.4	383.6	698.1	**212.7**	856.6
c6nidw-i	414.4	267.1	734.7	412.5	244.5	**209.7**	540.1	710.3
c7b	**29.4**	167.2	76.3	58.4	34.6	43.9	63.9	435.5
c7b-i	101.4	54.2	68.1	52.0	**49.5**	93.2	293.4	154.5
c9	**10.8**	51.2	11.4	32.8	11.8	21.0	44.1	83.1
c9nidw-s	**122.5**	625.2	246.9	864.8	287.2	446.7	952.6	285.2
g10b	385.3	388.8	446.0	183.6	**106.5**	225.6	291.2	182.5
g10id	736.0	350.7	524.0	723.9	98.3	**92.0**	190.6	188.4
g7nidw	119.4	24.8	78.3	67.3	**13.5**	17.2	63.6	37.8
Total speedup:		1.0x	0.9x	0.8x	2.1x	1.4x	1.0x	0.9x
Arithmetic average speedup:		1.8x	1.0x	1.1x	2.8x	2.3x	1.3x	1.4x
Harmonic average speedup:		0.5x	0.8x	0.6x	1.2x	0.9x	0.5x	0.3x

Table 4. *Incremental BMC on original and minimized AIG.* The above problems all contain bugs. Runtimes are given for performing incremental BMC upto the shortest counter example. In the columns to the right of the arrows, the design has been minimized by DAG-aware rewriting before unrolling it. The node count is the number of ANDs in the design. Note that in this scheme, there can be no cross-timeframe simplifications. The experiment confirms the claim in [2] of the applicability of DAG-aware circuit comparession to formal verification. The original paper only listed compression ratios and did not include runtimes.

Problem	Nodes before and after minimization	BMC runtimes before and after minimization
Cdn1	3,527 → 949	37.8 s → 9.6 s
Cdn2	7,918 → 3,126	17.5 s → 0.8 s
Cdn3.1	84,718 → 28,637	607.1 s → 275.3 s
Cdn3.3	84,698 → 28,611	>1 h → 1823.7 s
Cdn4	2,936 → 1,538	>1 h → >1 h
nusmv:tcas$_5$	2,661 → 1,975	9.11 s → 2.27 s
nusmv:tcas$_6$	2,656 → 1,965	4.12 s → 0.67 s
texas.parsesys$_1$	11,860 → 939	0.64 s → 0.03 s
texas.two-proc$_2$	791 → 335	0.23 s → 0.01 s
vis.eisenberg	720 → 306	1.63 s → 2.01 s

Table 5. *Comparing CNF generation through standard technology mapping and technology mapping with the cut cost function adapted for SAT.* In the adapted CNF generation based on technology mapping (**righthand side of arrows**), the area of a LUT is defined as the number of clauses needed to represent its boolean function. In the standard technology mapping (**lefthand side of arrows**), each LUT has unit area "1". In both cases, the mapped design is translated to CNF by the method described in section 6.4, which introduces one variable for each LUT in the mapping. The standard technology mapping minimizes the number of LUTs, and hence will have a lower number of introduced variables. From the table it is clear that using the number of clauses as the area of a LUT gives significantly fewer clauses, and also reduces SAT runtimes.

	Technology Mapping for CNF		
Problem	#clauses	#vars	SAT-time
Cdn1-70u	62 k → 54 k	12 k → 15 k	6.6 s → 4.1 s
Cdn1-71s	64 k → 55 k	13 k → 15 k	6.6 s → 3.1 s
Cdn2-154u	327 k → 312 k	58 k → 77 k	23.3 s → 34.4 s
Cdn2-155s	333 k → 318 k	58 k → 78 k	21.4 s → 50.6 s
Cdn3.1-18u	1990 k → 905 k	145 k → 248 k	125.9 s → 78.8 s
Cdn3.1-19s	2147 k → 977 k	156 k → 267 k	161.2 s → 137.1 s
Cdn3.2-19u	2146 k → 977 k	156 k → 266 k	189.9 s → 73.7 s
Cdn3.2-20s	2302 k → 1049 k	167 k → 285 k	501.6 s → 313.5 s
Cdn3.3-19u	2147 k → 977 k	156 k → 267 k	136.4 s → 107.6 s
Cdn3.3-20s	2302 k → 1049 k	167 k → 285 k	311.7 s → 236.9 s

References

1. A. Biere. **AIGER** (AIGER is a format, library and set of utilities for And-Inverter Graphs (AIGs)). *http://fmv.jku.at/aiger/*.
2. P. Bjesse and A. Boralv. **DAG-Aware Circuit Compression For Formal Verification**. In *Proc. ICCAD'04*, 2004.
3. D. Chen and J. Cong. **DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs.** In *ICCAD*, pages 752–759, 2004.
4. R. Drechsler. **Using Synthesis Techniques in SAT Solvers**. *Technical Report, Intitute of Computer Schience, Unversity of Bremen, 28359 Bremen, Germany*, 2004.
5. N. Een. http://www.cs.chalmers.se/~een/SAT-2007.
6. N. Een and A. Biere. **Effective Preprocessing in SAT through Variable and Clause Elimination**. In *Proc. of Theory and Applications of Satisfiability Testing, 8th International Conference (SAT'2005)*, volume 3569 of *LNCS*, 2005.
7. N. Een and N. Sörensson. **Translating Pseudo-Boolean Constraints into SAT**. In *Journal on Satisfiability, Boolean Modelling and Computation (JSAT)*, volume 2 of *IOS Press*, 2006.
8. B. L. S. Group. **ABC: A System for Sequential Synthesis and Verification**. *http://www.eecs.berkeley.edu/~alanmi/abc/*.
9. P. Jackson and D. Sheridan. **Clause Form Conversions for Boolean Circuits**. In *Theory and Appl. of Sat. Testing, 7th Int. Conf. (SAT'04)*, volume 3542 of *LNCS, Springer*, 2004.

10. S. Minato. **Fast Generation of Irredundant Sum-Of-Products Forms from Binary Decision Diagrams**. In *Proc. SASIMI'92*.
11. A. Mishchenko, S. Chatterjee, and R. Brayton. **DAG-aware AIG rewriting: A fresh look at combinational logic synthesis**. In *Proc. DAC'06*, pages 532–536, 2006.
12. A. Mishchenko, S. Chatterjee, and R. Brayton. **Improvements to Technology Mapping for LUT-based FPGAs**. volume 26:2, pages 240–253, February 2007.
13. C. Sinz. **SAT-Race 2006 Benchmark Set**. *http://fmv.jku.at/sat-race-2006/*.
14. G. Tseitin. **On the complexity of derivation in propositional calculus**. *Studies in Constr. Math. and Math. Logic*, 1968.
15. M. N. Velev. **Efficient Translation of Boolean Formulas to CNF in Formal Verification of Microprocessors**. *Proc. of Conf. on Asia South Pacific Design Aut. (ASP-DAC)*, 2004.
16. E. Zarpas. **Benchmarking SAT Solvers for Bounded Model Checking**. In *Proc. SAT'05*, number 3569 in LNCS. Springer-Verlag, 2005.

Towards a Better Understanding of the Functionality of a Conflict-Driven SAT Solver*

Nachum Dershowitz[1,3], Ziyad Hanna[2], and Alexander Nadel[1,2]

[1] School of Computer Science, Tel Aviv University, Ramat Aviv, Israel
{nachumd,ale1}@tau.ac.il
[2] Design Technology Solutions Group, Intel Corporation, Haifa, Israel
{ziyad.hanna,alexander.nadel}@intel.com
[3] Microsoft Research, Redmond, WA

Abstract. We show that modern conflict-driven SAT solvers implicitly build and prune a decision tree whose nodes are associated with flipped variables. Practical usefulness of conflict-driven learning schemes, like 1UIP or *All*UIP, depends on their ability to guide the solver towards refutations associated with compact decision trees. We propose an enhancement of 1UIP that is empirically helpful for real-world industrial benchmarks.

1 Introduction

Modern conflict-driven backtrack-search SAT solvers are widely used in applications in academia and industry. Each invocation can be associated with a decision tree, and tree pruning is a commonly used, intuitive concept for developing and analyzing enhancements. But, since the introduction of Conflict-Directed Backjumping (CDB) [4], it has become unclear how to characterize the decision tree built in the process. The main difficulty arises from the fact that a CDB-based solver may flip values of implied variables, rather than decision variables. Also, it may skip decision levels when backtracking. As a result of this vagueness, modern solvers are more commonly understood as resolution engines, using decision-tree construction as a heuristic, rather than as algorithms constructing decision trees (e.g., [3]). Unfortunately, this provides little insight for reasoning about the behavior of learning schemes and for developing new ones. Witness the statement [5]: "The effectiveness of certain ... schemes can only be determined by empirical data for the entire solution process".

We propose a framework that allows one to reason about a CDB-based solver as a decision-tree construction based engine. We rely on the following hypothesis: nodes in the decision tree, implicitly constructed by a CDB-based solver, are associated with flipped variables, rather than with initially picked decision

* This research was supported in part by the Israel Science Foundation (grant no. 250/05). The work of Alexander Nadel was carried out in partial fulfillment of the requirements for a Ph.D.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 287–293, 2007.

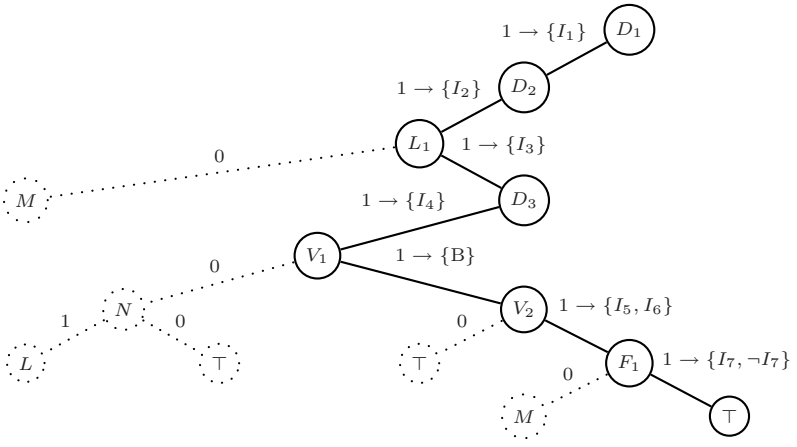


Fig. 1. Snapshot of a CDB-based solver run. The solid rightmost path is the current assignment stack. There are three decision levels. Each flipped variable is associated with a left decision subtree, denoted by dotted parts. Nodes correspond to decision or flipped variables and edges are marked with the Boolean values assigned to these variables and, optionally, with implied literals.

variables. This approach allows us to explain why 1UIP [1] is empirically advantageous over other schemes (cf. [5,3]). It also suggests a practically useful enhancement, called "local conflict clause recording".

2 Implicit Decision-Tree Construction and Pruning

An *asserting conflict clause* is a conflict clause containing the negation of one and only one literal, called a *pivot literal*, assigned at the last decision level. The 1UIP [1], 2UIP [5] and *All*UIP [5] clauses are all asserting. After the pivot variable is flipped, it is called a *flipped variable*. The *parent clause* of an implied literal A, denoted $Par(A)$, is the clause where the value of A is implied.

Decision-tree construction for plain backtracking can be understood as adding a new node to the tree, labeled with a decision variable B, assigned value $\sigma = Val(B)$, and a new left edge, labeled σ, upon each decision. The left subtree of B, denoted $LTree(B)$, is constructed recursively. When the solver backtracks to B and flips $Val(B)$, the tree is updated with a new right edge, labeled $\neg\sigma$, and a right subtree is constructed.

In our view, a CDB-based solver maintains a forest of left subtrees. Every flipped variable is associated with a left subtree. The forest is merged into one tree, comprising a refutation trace of the whole formula, only after the last conflict. Upon conflict, when a pivot variable B is flipped, its left decision subtree is constructed by merging left subtrees of a subset of flipped variables, assigned after B. Suppose the solver is in a conflict situation, the conflicting clause is γ and the decision level is k. We call a flipped variable that belongs to level k an *lf-variable*, and a flipped variable that belongs to levels lower than k an

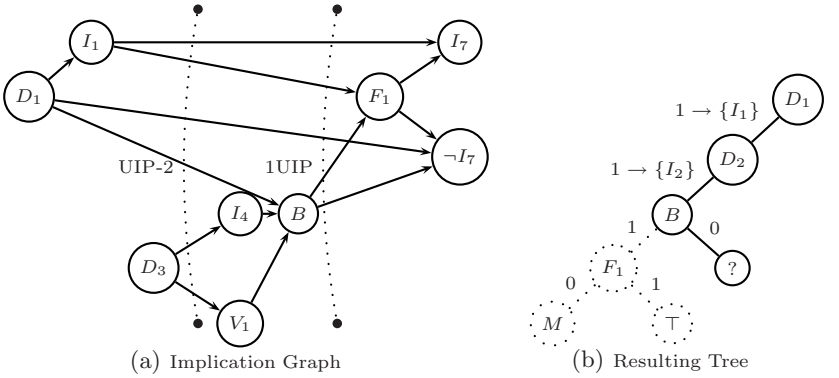


Fig. 2. Implication graph and decision tree for Fig. 1 with 1UIP and UIP-2 cuts and the resulting tree after applying Algorithm 1 and conflict-driven backjumping for 1UIP scheme

lu-variable. An lf-variable is *active* if it is connected to γ and is dominated by B in the implication graph. In our example (Fig. 1 and Fig 2(a)), the only active lf-variable is F_1. Lf-variable V_1 is not dominated by B. Lf-variable V_2 is not connected to the conflicting variable. Thus, both V_1 and V_2 are inactive.

Algorithm 1. *On conflict, returns LTree*(B) *of the pivot variable* B

1: Let $F_1 \dots F_n$ be active lf-variables. Suppose $LTree(F_{n+1})$ and $Tree(F_1)$ are leaves.
2: **for** $i := n$ **downto** 1 **do**
3: $\quad Tree(F_i) := TNewTree(F_i; \neg Val(F_i); LTree(F_i); Tree(F_{i+1}))$
4: **return** $Tree(F_1)$

Algorithm 1 constructs the left decision subtree of a pivot variable B. A recursive function *TNewTree* is invoked. It receives four parameters: (1) root variable; (2) first value of the root variable; (3) left subtree; and (4) right subtree. See Fig. 2(b) for the result of applying Algorithm 1 and conflict-driven backjumping for 1UIP scheme in our example.

Applying Algorithm 1 allows a CDB-based solver to *skip* some flipped variables. Skipping a flipped variable means excluding its left subtree from the final decision tree characterizing the run of a solver. Skipped variables fall into three categories: (1) lu-variables, skipped during backtracking (L_1 in our example); (2) inactive lf-variables, connected to the conflicting clause vertices, but not dominated by the pivot variable (V_1 in our example); (3) inactive lf-variables, not connected to the conflicting clause vertices (V_2).

We distinguish between two types of decision-tree pruning: *backward tree pruning* is carried out upon conflict detection by skipping existing subtrees; *forward tree pruning* is performed by recording conflict clauses useful in terms of frequent participation in Boolean constraint propagation (BCP) during the subsequent search. Algorithm 1 carries out backward tree pruning implicitly by

not including the left decision subtrees of inactive lf-variables in the left decision subtree of the pivot variable. To the best of our knowledge, this kind of decision-tree pruning has not been highlighted in the literature. A more prominent kind of backward tree pruning is carried out by the solver while backtracking non-chronologically [4]. We underscore the fact that the effectiveness of this kind of pruning depends on the size of the left decision subtrees of skipped flipped variables, rather than on the number of skipped decision levels, as usually presumed.

3 Usefulness of Conflict-Clause Recording Schemes

The *UIP-2* scheme for conflict learning takes UIP number 2 of the last decision level as the pivot variable. We compared the best known scheme, 1UIP [1], with *All*UIP [5] and UIP-2, which we feel are representative enough to explain the advantages of 1UIP over other schemes, too. (We do not discuss conflict clause minimization due to space restrictions.)

Choosing the first UIP, rather then UIP number 2 of the last decision level, is optimal for backward pruning. Indeed, the first UIP is the closest to the conflict; thus it tends to dominate fewer lf-variables. Also, the first UIP allows backtracking to the highest possible decision level, maximizing the number of uf-variables skipped during backtracking.

Why is 1UIP better than *All*UIP? Replacing literals of other decision levels by their dominator does not impact backward tree pruning. Indeed, the number of inactive lf-variables and the backtrack level remain the same. We claim that 1UIP clauses tend to contribute more to BCP than *All*UIP clauses, so are more useful for forward pruning. Let B be the pivot variable and k the decision level at the moment of a conflict. Denote by $Fr^+(B)$ the fraction of the conflict clauses that contain the variable B out of all conflict clauses recorded since B was last assigned. The key observation, confirmed empirically in Sect. 5, is that $Fr^+(B)$ tends to be much higher for *All*UIP than for 1UIP. Indeed, 1UIP conflict clauses

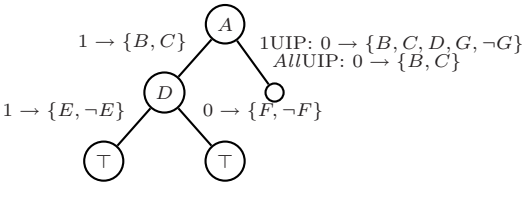


Fig. 3. Example of superiority of 1UIP over *All*UIP. Suppose we invoke a CDB-based SAT solver on an input formula $(A \vee D \vee G) \wedge (A \vee D \vee \neg G) \wedge (A \vee C) \wedge (A \vee B) \wedge (\neg A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee \neg C \vee \neg D \vee E) \wedge (\neg B \vee \neg C \vee \neg D \vee \neg E) \wedge (\neg A \vee D \vee F) \wedge (\neg A \vee D \vee \neg F)$. The solver first picks the literal A, propagates its value, then picks D, propagates and encounters a conflict. The 1UIP clause is $\neg B \vee \neg C \vee \neg D$; the *All*UIP clause is $\neg A \vee \neg D$. After flipping D, both the *All*UIP and the 1UIP conflict clauses are $\neg A$. After propagating, 1UIP would yield a conflict, meaning that the formula is unsatisfiable. In contrast, *All*UIP would not result in a conflict, since all previously recorded conflict clauses are satisfied.

tend to contain literals implied from B at k, rather than B itself. *All*UIP clauses tend to contain B, since B dominates all the literals at k. Hence, after flipping B, more of the *All*UIP conflict clauses, recorded before the flip, will be satisfied and will not contribute to BCP (compared with 1UIP conflict clauses). See Fig. 3 for an example.

4 Local Conflict-Clause Recording

A *Local Conflict-Clause (LCC)* is a non-asserting conflict clause, recorded in addition to the 1UIP conflict clause if the last decision level contains some active lf-variables. To record it, the last active lf-variable is considered to be a decision variable, defining a new decision level. An LCC is the 1UIP clause with respect to this new decision level.

A clause α is *inconsistent* with a decision-tree path P if α contains the negation of one of the literals of P. Consider a conflict situation, with pivot variable B and active lf-variables $F_1, F_2, \ldots, F_n$. Suppose the leftmost path of $LTree(B)$ is $P_1 = (G_1, \ldots, G_l)$. The rightmost path of $LTree(B)$ must be $P_f = (F_1, \ldots, F_n)$. The key observation is that there is an asymmetry between P_1 and P_f in that P_1 tends to be inconsistent with more clauses than P_f. Indeed, each of the clauses $Par(G_i)$ is inconsistent with P_1, since it must contain $\neg G_i$. This is not the case with P_f. It is not guaranteed that there exist clauses containing $\neg F_j$, since parent clauses of F_j's contain F_j rather than $\neg F_j$. Denote the number of left edges in a path by $\ell(P)$. An arbitrary path P in $LTree(B)$ is guaranteed to be inconsistent with at least $\ell(P)$ clauses. In general, the greater $\ell(P)$, the greater the chance is that there will be aggressive propagation, once the literals of P are assigned.

The main goal of adding LCCs is to improve forward tree pruning when literals, corresponding to a path with small $\ell(P)$, are assigned. In addition, LCCs tend to contribute more to BCP than 1UIP clauses immediately after flipping the pivot variable. Indeed, after flipping the pivot variable, the 1UIP clause is always satisfied, whereas the local conflict clause may contribute to BCP, since it may not contain the pivot variable.

5 Experimental Results

We implemented 1UIP, UIP-2 and *All*UIP within the industrial CDB-based solver, Eureka [2] (but without decision-stack shrinking). All experiments were carried out on a machine with 4GB memory and two Intel Xeon CPU 3.06 processors. We used instances from 11 well-known industrial benchmark families. These three schemes are compared in Table 1 on 8 instances.

The main conclusions of our experiments are: (1) 1UIP is indeed more powerful and robust than other schemes. It is always faster than UIP-2, and outperforms *All*UIP by orders of magnitude on 4 instances, appearing in the left column of Table 1. (2) Fr^+ is double for *All*UIP than for 1UIP. This explains 1UIP's superiority over *All*UIP by confirming the hypothesis of Sect. 3. (3) Of

Table 1. Comparing 1UIP, UIP-2 and *All*UIP on selected instances. The rows display: (Tm) execution time in seconds; (Con) number of conflicts; (Fr^+) average Fr^+; (NSk) average number of decision-tree nodes skipped per conflict.

Instance	Res	1UIP	UIP-2	*All*UIP	Instance	Res	1UIP	UIP-2	*All*UIP
4pipe	Tm	51	148	11930	*longmult10*	Tm	485	513	590
	Con	101277	308946	29985706		Con	237814	261669	379737
	Fr^+	0.41	0.38	0.83		Fr^+	0.37	0.34	0.84
	NSk	0.19	0.14	0.24		NSk	0.13	0.11	0.24
5pipe	Tm	50	347	> 14400	*longmult11*	Tm	559	756	690
	Con	85119	562304	28185547		Con	273200	346414	471626
	Fr^+	0.40	0.33	0.84		Fr^+	0.37	0.35	0.83
	NSk	0.18	0.14	0.21		NSk	0.14	0.11	0.25
8pipe_k	Tm	2426	> 14400	> 14400	*rotmul*	Tm	578	1186	992
	Con	1478419	10129202	13192438		Con	615314	1371339	1576324
	Fr^+	0.37	0.26	0.81		Fr^+	0.52	0.48	0.84
	NSk	0.21	0.13	0.19		NSk	0.16	0.13	0.27
9pipe_k	Tm	1493	> 14400	> 14400	*term1mul*	Tm	2173	5213	2975
	Con	640559	6040439	6548156		Con	1585135	3750774	3059096
	Fr^+	0.37	0.27	0.85		Fr^+	0.55	0.54	0.86
	NSk	0.20	0.16	0.20		NSk	0.15	0.11	0.26

Table 2. Effect of LCC recording (time is in sec.; t/o is the number of instances that timed out)

Family	Threshold	Default Time	Default t/o	Def. + LCC Time	Def. + LCC t/o
sat04_ind_maris03_gripper_sat	3 hours	2238	0	986	0
sat04_ind_goldberg03_hard_eq_check	3 hours	30336	2	15353	0
sat04_ind_maris03_gripper_unsat	4 hours	30135	4	17842	2
velev_fvp_unsat.3.0	3 hours	18199	2	10928	2
velev_fvp_sat.3.0	3 hours	9041	0	7155	0
velev_vliw_sat_2.0	3 hours	5970	0	4715	0
barrel	3 hours	260	0	226	0
velev_pipe_unsat_1.0	3 hours	15880	0	13094	0
velev_vliw_unsat_4.0	3 hours	17260	0	14810	0
longmult	3 hours	5413	0	5076	0
velev_vliw_sat_4.0	3 hours	5116	0	6882	0

all schemes, UIP-2 skips the fewest nodes/flipped variables. Additional empirical findings, omitted here, show that this happens mainly due to the fact that there are fewer inactive lf-variables not dominated by the pivot variable in the implication graph. This agrees with the theoretical analysis in Sect. 3. (4) Surprisingly, *All*UIP allows one to skip more nodes and flipped variables than 1UIP on some examples. We found that it happens mainly due to the fact that many lf-variables are not connected to the conflicting clause for *All*UIP. According to the analysis in Sect. 3, the number of skipped nodes and variables should be about the same for both schemes. This expected behavior is indeed observed on the 4 instances of the left column of Table 1, where *All*UIP is outperformed by several orders of magnitude. Studying the reasons for the unexpected behavior on the other 4 instances, where the gap between 1UIP and *All*UIP is not large, is left for future research.

Table 2 shows the effect on 11 families of local conflict-clause recording within the default version of Eureka. The technique is helpful overall on 10 of them.

Accordingly, LCC recording can be recommended as a default strategy for modern CDB-based solvers.

References

1. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC'01*, pages 530–535, 2001.
2. A. Nadel, M. Gordon, A. Palti, and Z. Hanna. Eureka-2006 SAT solver. `http://fmv.jku.at/sat-race-2006/descriptions/4-Eureka.pdf`.
3. L. O. Ryan. Efficient algorithms for clause learning SAT solvers. Master's thesis, Simon Fraser University, Burnaby, Canada, 2004.
4. J. P. M. Silva and K. A. Sakallah. GRASP—a new search algorithm for satisfiability. In *ICCAD'96*, pages 220–227. IEEE Computer Society, 1996.
5. L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *ICCAD'01*, pages 279–285. IEEE Press, 2001.

A Lightweight Component Caching Scheme for Satisfiability Solvers

Knot Pipatsrisawat and Adnan Darwiche

Computer Science Department
University of California, Los Angeles
{thammakn,darwiche}@cs.ucla.edu

Abstract. We introduce in this paper a lightweight technique for reducing work repetition caused by non–chronological backtracking commonly practiced by DPLL–based SAT solvers. The presented technique can be viewed as a partial component caching scheme. Empirical evaluation of the technique reveals significant improvements on a broad range of industrial instances.

1 Introduction

As a DPLL–based SAT solver makes decisions, the knowledge base gets simplified due to Boolean constraint propagation. This simplification may be substantial enough to disconnect the knowledge base into independent components[1]. Knowledge about independent components could reduce the amount of work done by a solver. However, precise component analysis is prohibitively expensive for SAT solving in general, although some solvers have incorporated static component analysis in the preprocessing phase [1,9,6].

The lack of dynamic component analysis is made worse by the use of non–chronological backtracking, because it may cause solvers to erase assignments that are not related to the conflict. In the worst case, erased assignments may contain solutions to independent components. As a result, the solver may need to solve some components multiple times. This problem has already been observed and solutions have been proposed in [8,4]. Nevertheless, the proposed solutions seem to offer limited improvements on real–world instances.

We address this particular problem in this paper and provide two contributions. First, an analytic and empirical analysis that substantiates the observations about work repetition in modern SAT solvers that use non–chronological backtracking. Second, a low–overhead technique that helps reduce work repetition in such solvers.

The rest of this paper is structured as follows. In Section 2, we describes precisely the above problem. An empirical study that further exposes and quantifies the problem is presented in Section 3. A solution is proposed in Section 4 and is evaluated in Section 5. Section 6 discusses related work and we conclude in Section 7.

[1] A component is defined as a set of clauses. Two components are independent if they share no variable.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 294–299, 2007.

2 Losing Work with Non–chronological Backtracking

The use of non–chronological backtracking in SAT and CSP allows solvers to better focus on fixing the cause of the conflict [18,2,11]. The most common non–chronological backtracking scheme used by SAT solvers today, called far–backtracking [16], is based on generating *asserting clauses* [20]. This approach involves undoing the assignments from the point of conflict up to (not including) the assertion level. Although the results and analysis we present in this paper can be adapted to work for non–chronological backtracking in general, we stay focused on far–backtracking as it is the most common in modern SAT solvers.

One caveat on this backtracking scheme is that all decisions and implied variable assignments made between the level of the conflict and the assertion level are effectively erased by backtracking. As we shall see next, these assignments may contain solutions of sub–problems (components) and would be lost in the backtracking process, requiring their rediscovery at a later stage in the search.

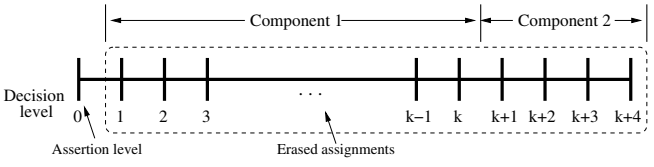


Fig. 1. Erased assignments due to a backtrack

To consider a dramatic example of this phenomena, examine Figure 1 in which the solver has solved a component using the first k decision levels. After several decisions on a second component, the solver runs into a conflict and derives a unit learned clause. The assertion level in this case will be level 0, leading the solver to erase all assignments, assert the unit clause, and restart the search process all over. After the learned clause is asserted, the solver will continue looking for solutions for both components, as it did not save the solution it previously found. This can lead to great inefficiency as the solver may end up solving some components multiple times.

We will provide a more realistic, yet still somewhat synthetic, empirical study in the next section to quantify further this potential inefficiency.

3 An Empirical Study

To illustrate the extent of work repetition, we artificially generated instances that would cause work repetition in conventional SAT solvers. Each instance was generated by merging four identical copies of a satisfiable instance. These bigger instances will be referred to as replicated instances throughout this paper.

Table 1 reports the results of this initial experiment, conducted using MiniSat [7], on a computer with Intel Pentium 3.4GHz processor and 2GB of RAM.

Table 1. Runtime (in seconds) of MiniSat with and without progress saving. (*) indicates insufficient memory. The ratio columns show approximate ratios of runtime on replicated over original instances.

Instance Name	Runtime					
	MiniSat			MiniSat with ps		
	Original	Replicated	Ratio	Original	Replicated	Ratio
vmpc_21.renamed-as.sat05-1923	6.01	731.98	122	1.5	21.58	14
vmpc_21.shuffled-as.sat05-1955	0.48	59.37	124	1.25	26.01	21
vmpc_23.renamed-as.sat05-1927	39.19	3202.67	82	2.4	28.61	12
vange-color-54	28.26	4624.42	163	4.24	96.34	23
velev-fvp-sat-3.0-12	6.70	>200*	>29	4.07	41.87	10
ibm_19_rule_SAT_dat.k30	7.91	209.73	26	6.47	28.6	4.4
ibm_21_rule_SAT_dat.k35	8.87	819.00	92	5.15	44.12	8.6

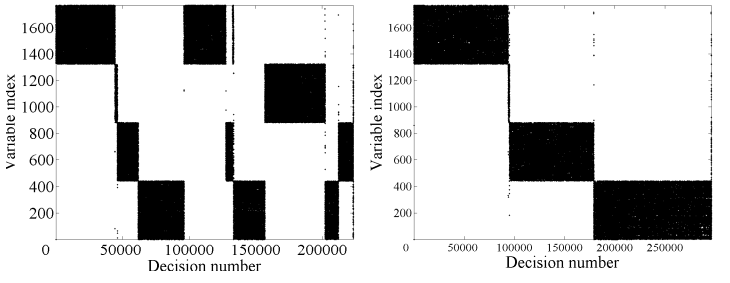


Fig. 2. Decision behavior on a replicated instance of MiniSat (left) and MiniSat with progress saving (right). Both x-axes represent the chronological order of decisions.

The table reports runtime of MiniSat on each original and replicated instance. Let us first consider the first three runtime columns. The remaining columns will be discussed in the next section. According to this table, MiniSat can be more than two orders of magnitude slower on replicated instances, even though a replicated instance contains four identical copies of the original instance.

Further investigation on these instances reveals the source of inefficiency. In the next experiment, we plot indices of decision variables in chronological order. The left plot in Figure 2 shows such plot based on running MiniSat on the replicated instance of vmpc_21.shuffled-as.sat05-1955. Variable indices in the replicated instance range from 1 to 1764 (4×441 original variables). Each independent component in the instance occupies a contiguous range of variable indices. Each dark band in this plot indicates the solver's attempt to solve a component. We can see in this plot that MiniSat ended up solving all components multiple times. Most of the attempts to re-solve a component take non-trivial amount of work, as illustrated by the width of each band. This clearly illustrates that work repetition is responsible for a fair amount of the disproportionate increase in runtime of the solver on the replicated instances. Further experiments revealed that similar behavior persisted on other instance pairs as well.[2]

[2] More experimental results are available in an extended version of the paper at http://reasoning.cs.ucla.edu/publications.html

Table 2. Runtime of MiniSat with and without progress saving. * The solvers' runtime for each suite is calculated from instances solved by both versions.

Suite	Instance Count	Runtime*		# Solved	
		MiniSat	P. Saving	MiniSat	P. Saving
fvp sat 3.0	20	1134.027	45.59	10	20
grieu 05	32	5069.342	1789.555	16	19
IBM 2004 1_11	19	4422.805	1623.339	14	18
IBM 2004 1_14	19	329.039	194.078	19	19
manol pipe	31	5050.709	5247.547	30	31
pipe sat 1.0	10	92.108	973.436	6	8
liveness sat 1.0	10	114.576	888.857	5	6
vliw sat 2.0	9	59.312	887.14	5	5
narain 05	10	194.998	249.998	5	5
Total	160	16466.916	11899.54	110	131

4 A Lightweight Caching Scheme

The solution we are proposing for this problem is simple and can be thought of as a lightweight partial component caching technique. Since far–backtracking could erase partial solutions, we simply save them. This technique, which we refer to as *progress saving*, requires keeping an additional array of literals, called the saved–literal array. Every time the solver performs a backtrack and erases assignments, each erased assignment is saved in the this array. Now, any time the solver decides to branch on variable v, it uses the saved literal, if one exists. Otherwise, the solver uses the default phase selection heuristic. Note that this technique would fit nicely on any Chaff-like solver implementation.

We integrated progress saving into MiniSat for the purpose of evaluation[3]. In this integration, the variable ordering heuristic needs not be changed. Now, consider the last three columns of Table 1, in which the runtimes of MiniSat with progress saving on original and replicated instances are compared. In all cases, there are significant improvements in runtime on replicated instances.

Furthermore, the decision behavior of MiniSat with progress saving on the replicated instance of vmpc_21.shuffled-as.sat05-1955 is shown on the right of Figure 2. This plot indicates a decease in work repetition. Though previously solved components are still revisited (thin strips of dots after dark bands), their solutions are almost immediately found, because of the saved literals.

5 Experimental Results

We now evaluate progress saving on a set of 1251 industrial benchmarks drawn from the SAT'05 competition [17] and [19,10]. All experiments were performed on a Pentium 4, 3.8 GHz and 2GB RAM, with time limit of 1800 seconds.

Table 2 reports runtime on 160 instances selected from the total 1251 instances considered. According to this table, progress saving solves 21 more instances than MiniSat, improves the overall running time on those instance solved by both solvers, yet leads to worse running time on some of the instances.

[3] Progress saving was originally introduced in RSat [12,14].

Figure 3 provides more comprehensive evidence on the effectiveness of progress saving as it considers all 1251 instances discussed above. On the left, we compare three versions of MiniSat (different phase selection heuristics) to MiniSat with its default heuristic augmented with progress saving. The x-axis lists the number of solved instances for a given cutoff time (y-axis), showing that progress saving dominates all three versions of MiniSat. On the right, we show a head-to-head runtime comparison between MiniSat and MiniSat with progress saving. Note that both axes here are in log-scale. This figure, which also depicts the best linear fit, provides further evidence on the effectiveness of progress saving.

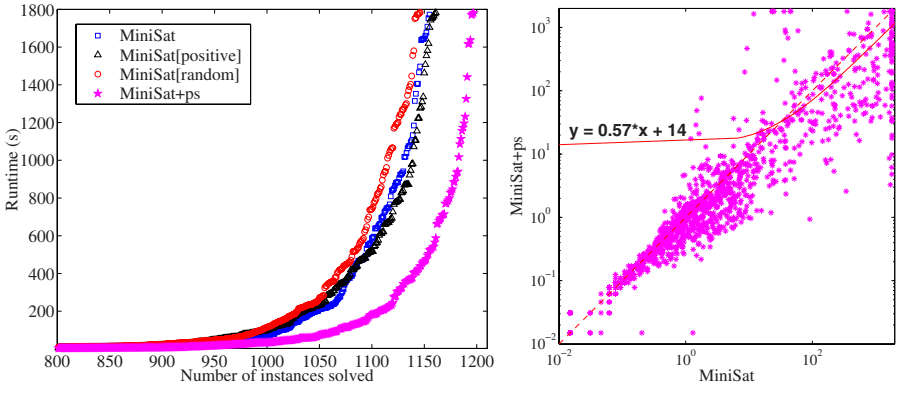


Fig. 3. Comparing three versions of MiniSat (different phase selection heuristics) to MiniSat with progress saving. The default phase selection heuristic of MiniSat splits on negative literals. We also consider splitting on positive and randomly chosen literals.

6 Related Work

Model counters and knowledge compilers have proven to benefit greatly from dynamic and semi–dynamic component analysis performed during the search [13,15,5,3]. These techniques are usually too expensive to apply to SAT solving. Ginsberg addressed a very similar problem in the context of CSP [8]. The author proposed a new backtracking scheme called *dynamic backtracking*. This approach is superficially similar to ours. However, it may cause the search space after backtracking to become overly constrained, as pointed out by the author. Moreover, it would require a careful modification of the contemporary SAT framework to make it work as intended. Neither is the case for our solution.

Biere and Sinz showed that independent components do exist in some real-world SAT instances and proposed an efficient method to take advantage of the structure [4]. However, their approach is semi–dynamic, as it only considers permanent decompositions that occur in the absence of any decision. While improvements on artificially–generated instances were reported, similar gains did not materialized in their experiment on real–world instances.

7 Conclusion

We studied an inefficiency introduced by the conventional backtracking scheme of modern SAT solvers. We then proposed a low–overhead solution, called progress saving, that can be viewed as a component caching technique. The practicality of our solution is illustrated by experimental results, which show improvements on a wide range of problems when the technique is integrated into MiniSat.

References

1. Aloul, F., Markov, I., and Sakallah, K. Force: a fast and easy-to-implement variable-ordering heuristic. In *Proc. of the 13th ACM Great Lakes Symposium on VLSI 2003. pp. 116-119.* (2003).
2. Bayardo, R. J. J., and Schrag, R. C. Using CSP look-back techniques to solve real-world SAT instances. In *AAAI'97* (Providence, Rhode Island), pp. 203–208.
3. Beame, P., Impagliazzo, R., Pitassi, T., and Segerlind, N. Memoization and dpll: Formula caching proof systems. In *Proc. of 18th Annual IEEE Conf. on Computational Complexity, Aarhus, Denmark.* (2003).
4. Biere, A., and Sinz, C. Decomposing sat problems into connected components. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT) 2* (2006).
5. Darwiche, A. New advances in compiling CNF to decomposable negational normal form. In *Proc. of European Conference on AI.* (2004).
6. Durairaj, V., and Kalla., P. Variable ordering for efficient sat search by analyzing constraint-variable dependencies. In *SAT'05* (August 2005).
7. Eén, N., and Sörensson, N. An extensible sat-solver. In *SAT'03* (2003).
8. Ginsberg, M. L. Dynamic backtracking. *Jrnl of Artf. Intel. Resrh. 1* (1993).
9. Huang, J., and Darwiche, A. A structure-based variable ordering heuristic for sat. In *(IJCAI'03)* (2003), pp. 1167–1172.
10. IBM. Ibm formal verification benchmark library. http://www.research.ibm.com /haifa/projects/verification/RB_Homepage/fvbenchmarks.html.
11. Marques-Silva, J. P., and Sakallah, K. A. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design* (1996), pp. 220–227.
12. Pipatsrisawat, K., and Darwiche, A. *SAT Solver Description: RSat.*
13. Roberto J. Bayardo, J., and Pehoushek, J. D. Counting models using connected components. In *Proc. of the 17th Natl. Conf. on AI.* (2000), AAAI Press / The MIT Press, pp. 157–162.
14. Rsat sat solver homepage. http://reasoning.cs.ucla.edu/rsat.
15. Sang, T., Bacchus, F., Beame, P., Kautz, H. A., and Pitassi, T. Combining component caching and clause learning for effective model counting. In *SAT'04.*
16. Sang, T., Beame, P., and Kautz, H. A. Heuristics for fast exact model counting. In *SAT* (2005), pp. 226–240.
17. SAT'05 Competition Homepage, http://www.satcompetition.org/2005/.
18. Stallman, R., and Sussman, G. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artf. Intel. 9* (1977).
19. Velev, M. N. Sat bnchmrk lib. www.miroslav-velev.com/sat_benchmarks.html.
20. Zhang, L., Madigan, C. F., Moskewicz, M. W., and Malik, S. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD* (2001), pp. 279–285.

Minimum 2CNF Resolution Refutations in Polynomial Time

Joshua Buresh-Oppenheim and David Mitchell

Simon Fraser University
jburesho@cs.sfu.ca, mitchell@cs.sfu.ca

Abstract. We present an algorithm for finding a smallest Resolution refutation of any 2CNF in polynomial time.

1 Introduction

The problem of deciding satisfiability of propositional 2-CNF formulas (2-SAT), is an important tractable case of SAT. The first polynomial-time algorithm for 2-SAT was given by Cook [4]. Linear time algorithms were given by Even, Itai and Shamir [6] and, subsequently, Aspvall, Plass and Tarjan [2]. For an unsatisfiable formula, a small and simple certificate, or proof of unsatisfiability, may be interpreted as an explanation for its unsatisfiability. Such explanations are central in a number of applications. Cook's algorithm constructs a tree-like Resolution refutation of an unsatisfiable formula. The algorithm of [6], and a later algorithm by del Val [5], involve schemes for applying unit Resolution, and can easily be modified to output tree-like Resolution refutations. The algorithm of [2] provides a certificate in the form of a graph labelled with clauses which are easily seen to be an unsatisfiable subset of the given clauses. A tree-like Resolution refutation can easily be extracted from this graph.

In [3] we gave polytime algorithms for finding a smallest tree-like Resolution refutation and a smallest unsatisfiable subformula of an unsatisfiable 2CNF (the latter is itself an efficiently verifiable certificate since 2SAT is in linear time). Here we give a polytime algorithm for finding a smallest general Resolution refutation. All three algorithms are dynamic programming algorithms based on the implication graph associated with a 2CNF. The algorithm for finding a tree-like refutation runs in time $O(n^2m)$, where n is the number of underlying variables and m is the number of clauses, while the algorithm presented here for general refutations runs in time $O(n^6m)$. In [3] we showed that minimum tree-like Resolution refutations provide a 2-approximation of the smallest general Resolution refutation. Hence, in practice it may often be better to use the faster algorithm to obtain an approximation. Nonetheless, we consider solving the general case an interesting theoretical problem. In particular, we note the contrast with the case of Horn formulas, for which the size of the smallest Resolution refutation is NP-hard to determine, or even to approximate within any constant factor [1]. This difference is especially interesting in light of the similarities in standard algorithms for 2-SAT and Horn-SAT. One may observe that we make essential

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 300–313, 2007.

use of the symmetries in 2-CNFs derivations, as exhibited by the "dual" paths in the implication graph, that do not exist in the Horn case.

The algorithms for finding minimum-size certificates provided here and in [3] are more complicated, and of higher time-complexity, than the linear-time algorithms. However, in some applications the size and simplicity of the certificates provided may justify the extra cost. For example, if we have a very large formula and the certificate must be interpreted by a human user, or we have plenty of time to preprocess the formula and the certificate produced will be used repeatedly in the future, then finding a sublinear size certificate, even if it takes a relatively long time, may be better than finding a linear size certificate quickly. Another example is provided by certain abstraction-based model checking techniques in hardware and software verification. At each stage in a sequence of stages a certificate of unsatisfiablility of one formula is used in the creation of a new, larger formula. The size and complexity of the certificates produced is very important in the success of the overall process. In general, the formulas used in this application are not 2CNF formulas, but they often have a very large fraction of 2-clauses. We envision being able to take advantage of the methods for constructing minimum refutations of 2CNF formulas in developing more effective algorithms than currently available for this context.

2 Preliminaries

Throughout, let $\mathcal{C}$ be a collection of 2-clauses (that is, clauses with at most two literals) over an ordered set of variables $\{x_1, ..., x_n\}$. Say $|\mathcal{C}| = m$. As first suggested by [2], $\mathcal{C}$ can be represented as a directed graph $G_{\mathcal{C}}$ on $2n$ nodes, one for each literal. If $(a \vee b) \in \mathcal{C}$ for literals a, b, then the edges $(\bar{a}, b)$ and $(\bar{b}, a)$ appear in $G_{\mathcal{C}}$ (note that literals a and b can be the same). Both of these edges are labelled by the clause $(a \vee b)$. For an edge $e = (a, b)$, let $dual(e)$, the dual edge of e, be the edge $(\bar{b}, \bar{a})$. For literals a, b, define $\mathcal{P}_{ab}$ to be the set of all directed paths from a to b in $G_{\mathcal{C}}$. If c is also a literal, let $\mathcal{P}_{abc}$ be the set of all directed paths that start at a, end at c and visit b at some point. For $P_1 \in \mathcal{P}_{ab}$ and $P_2 \in \mathcal{P}_{bc}$, we denote by $P_1 \circ P_2 \in \mathcal{P}_{abc}$ the concatenation of the two paths. For a path $P = (e_1, ..., e_k) \in \mathcal{P}_{ab}$, let $dual(P) \in \mathcal{P}_{\bar{b}\bar{a}}$ be the path $(dual(e_k), ..., dual(e_1))$.

Proposition 1 ([2]). *$\mathcal{C}$ is unsatisfiable if and only if there is a variable x such that both $\mathcal{P}_{x\bar{x}}$ and $\mathcal{P}_{\bar{x}x}$ are not empty.*

Actually, note that for any literals a, b and variable x, a pair of paths $P_1 \in \mathcal{P}_{a\bar{a}x}$ and $P_2 \in \mathcal{P}_{b\bar{b}\bar{x}}$ are contradictory (for one thing, they imply the existence of a pair of paths such as those in the proposition). This motivates the following definition: Two paths P_1 and P_2 are called *end-contradictory* if there are literals a and b and a variable x (x, $\bar{x}$ need not be distinct from $\bar{a}$, $\bar{b}$) such that $P_1 \in \mathcal{P}_{a\bar{a}x}$ and $P_2 \in \mathcal{P}_{b\bar{b}\bar{x}}$.

We will be interested in finding such pairs of paths of a particularly simple form. First we will need to establish several definitions about directed paths in $G_{\mathcal{C}}$. Note that in $G_{\mathcal{C}}$ even a simple path may contain two edges with the same

clause label. Let $clauses(P)$ denote the set of clause-labels underlying the edges of a directed path P. We define $|P|$, the *size* of the path P, to be $|clauses(P)|$. In contrast, let $length(P)$ denote the length of P as a sequence. Call a path P *singular* if it does not contain two edges that have the same clause label. Therefore, a path P is singular if and only if $|P| = length(P)$. Given two paths P_1, P_2, let $\ell(P_1, P_2)$ denote the quantity $|clauses(P_1) \cup clauses(P_2)|$.

Definition 1. *Let $suf(P)$ be the maximal singular suffix of P. For a path $P \in \mathcal{P}_{a\bar{a}b}$ ($\bar{a}$ and b need not be distinct), let $extend(P)$ be the following path in $\mathcal{P}_{\bar{b}b}$: let P' be the portion of P that starts at the last occurrence of $\bar{a}$ and goes to the end. Then $extend(P)$ is $dual(P') \circ P$.*

Definition 2. *Given a path $P \in \mathcal{P}_{a\bar{a}b}$, let $sing(P)$ be the following operation: first let $P' = extend(P)$. Now, while there is a repeated edge in P', remove the segment of P' after the first occurrence of the edge through the second occurrence. When there is no longer a repeated edge, take the suffix of the resulting path.*

It is clear that $sing(P)$ is singular and that $clauses(sing(P)) \subseteq clauses(P)$. Also, if P_1 and P_2 are end-contradictory, then so are $sing(P_1)$ and $sing(P_2)$ and $\ell(sing(P_1), sing(P_2)) \leq \ell(P_1, P_2)$.

Definition 3. *Let P be a singular path that starts at literal a. Define $core(P)$ as the subpath of P that starts at a and ends at the first occurrence of $\bar{a}$ (or at the end of P if there is none).*

A *segment* of a path is a consecutive subsequence of the path's sequence. For two singular paths P_1 and P_2, a *primal shared segment* is a common segment. A *dual shared segment* of P_1 with respect to P_2 is a segment t of P_1 such that $dual(t)$ is a segment of P_2. A *shared segment* is either a primal or dual shared segment. For two disjoint segments s and t of P, say $s \prec_P t$ if s appears before t in P. For two singular paths P_1 and P_2, let $k(P_1, P_2)$ be the number of maximal shared segments (primal or dual) of P_1 and P_2.

We assume the reader is familiar with Resolution derivations. We simply mention that Resolution derivations can be viewed as DAGs whose nodes are the clauses in the derivation (we assume all occurrences of a particular clause are identified to one node). In a derivation of a single clause C, C is the only source and the sinks are the axioms used in the derivation. Each non-axiom clause has fanout two: it points to the two clauses whose resolvent it is. A Resolution refutation is a derivation of the empty clause Λ. The size of a derivation is the number of clauses (nodes) in it.

Proposition 2. *Any Resolution derivation of a single clause that uses ℓ axioms must have size at least $2\ell - 1$.*

Let $P \in \mathcal{P}_{ab}$. Let $IR(P)$ be the Input Resolution derivation that starts by resolving the clauses labelling the first two edges in P and then proceeds by resolving the latest derived clause with the clause labelling the next edge in the sequence P. This is a derivation of either $(\bar{a} \vee b)$ or simply (b) (if the path goes through literal $\bar{a}$). It is not hard to see that the size of the derivation $IR(P)$ is $2 \cdot length(P) - 1$.

3 Characterizing Minimum Resolution Refutations

Let π be a Resolution derivation from $\mathcal{C}$ that includes the clause $(\bar{a} \vee b)$. Then π defines a path in $G_{\mathcal{C}}$ from a to b (and from $\bar{b}$ to $\bar{a}$). The underlying edges of this path are exactly the elements of $\mathcal{C}$ that appear as sinks in π and are reachable from $(\bar{a} \vee b)$. More formally, we have the following definition:

Definition 4. *Let a, b be literals over distinct variables. Let π be a Resolution derivation containing $(\bar{a} \vee b)$. If $(\bar{a} \vee b)$ is a sink in π, then let $ResPath(\pi, (\bar{a} \vee b), a \rightarrow b)$ equal the edge (a, b), and let $ResPath(\pi, (\bar{a} \vee b), \bar{b} \rightarrow \bar{a})$ equal the edge $(\bar{b}, \bar{a})$. Otherwise, assume $(\bar{a} \vee b)$ has children $(\bar{a} \vee c)$ and $(\bar{c} \vee b)$, for some literal c, in π. Then set $ResPath(\pi, (\bar{a} \vee b), a \rightarrow b)$ to $ResPath(\pi, (\bar{a} \vee c), a \rightarrow c) \circ ResPath(\pi, (\bar{c} \vee b), c \rightarrow b)$. Set $ResPath(\pi, (\bar{a} \vee b), \bar{b} \rightarrow \bar{a})$ to $ResPath(\pi, (\bar{c} \vee b), \bar{b} \rightarrow \bar{c}) \circ ResPath(\pi, (\bar{a} \vee c), \bar{c} \rightarrow \bar{a})$. If the variable underlying a precedes the variable underlying b in the order of variables, then let $ResPath(\pi, (\bar{a} \vee b)) = ResPath(\pi, (\bar{a} \vee b), a \rightarrow b)$.*

Now assume that the clause (a) appears in some Resolution derivation π. Again, if (a) is a sink, let $ResPath(\pi, (a))$ be the edge $(\bar{a}, a)$. Otherwise, if the children of (a) are $(a \vee x)$ and $(a \vee \bar{x})$ for some variable x, then set $ResPath(\pi, (a))$ to $ResPath(\pi, (a \vee x), \bar{a} \rightarrow x) \circ ResPath(\pi, (a \vee \bar{x}), x \rightarrow a)$. Otherwise, if the children of (a) are $(a \vee b)$ and $(\bar{b})$ for some literal b, then set $ResPath(\pi, (a))$ to $ResPath(\pi, (\bar{b})) \circ ResPath(\pi, (a \vee b), \bar{b} \rightarrow a)$.

Finally, given a Resolution refutation π that ends by resolving (x) and $(\bar{x})$, let $ResPath(\pi)$ be the pair $(ResPath(\pi, x), ResPath(\pi, \bar{x}))$.

Notice that, for a Resolution refutation π, the pair of paths in $ResPath(\pi)$ are end-contraditory. This justifies our strategy of reducing the search for a minimum Resolution refutation to a search for a pair of end-contradictory paths that satisfy certain criteria.

Definition 4 demonstrates that there is a pretty deep correspondence between Resolution derivations over $\mathcal{C}$ and paths in $G_{\mathcal{C}}$. Will we exploit this correspondence heavily throughout, but here we pause to illustrate one salient aspect of it. Consider a fragment of a Resolution derivation π such as that in figure 1. Let $Q = ResPath(\pi, C)$. Then, going backwards along the main path in the derivation, each successive clause C_i corresponds to an extension of the segment Q, called Q_i. In particular, the resolution with each clause D_i extends Q_i either from its beginning or from its end.

Definition 5. *A* joint derivation *of two clauses $(\bar{a} \vee b)$ and $(\bar{c} \vee d)$ (again, $\bar{a}$, $\bar{c}$ need not be distinct from b, d) from $\mathcal{C}$ is a Resolution derivation that uses $\mathcal{C}$ as axioms and such that $(\bar{a} \vee b)$ and $(\bar{c} \vee d)$ appear in the derivation and are the only clauses with fanin 0.*

Definition 6. *Consider a joint derivation π of $(\bar{a} \vee b)$ and $(\bar{c} \vee d)$ from $\mathcal{C}$. A* shared clause *in this derivation is any clause C in π such that there are paths in π from $(\bar{a} \vee b)$ to C and from $(\bar{c} \vee d)$ to C, respectively. A* top-shared *clause is a shared clause C such that there is a path from $(\bar{c} \vee d)$ to C that contains no other shared clause.*

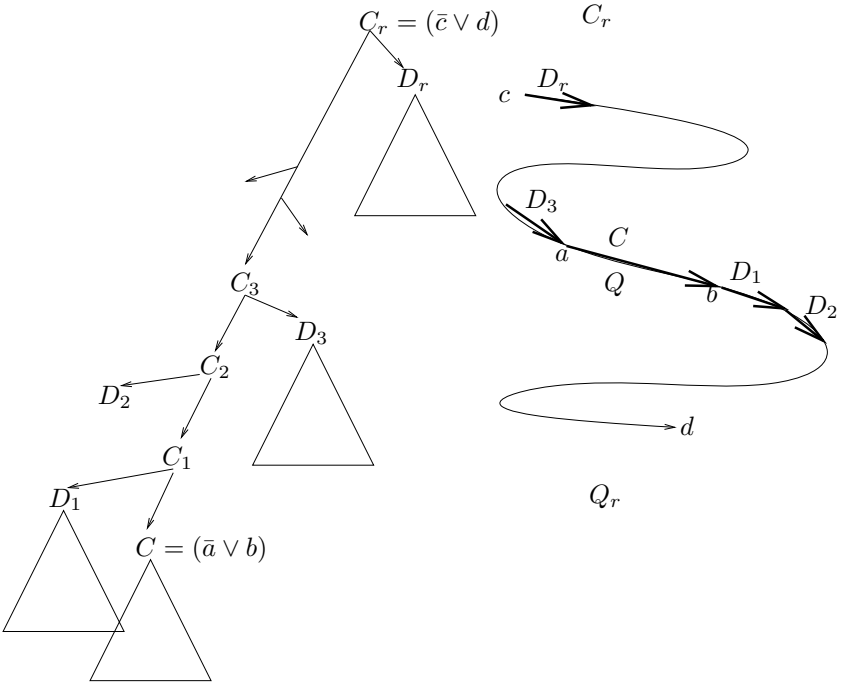


Fig. 1. Extending a path through Resolution

Lemma 1. *Let π be a joint derivation of $(\bar{a} \vee b)$ and $(\bar{c} \vee d)$ from $\mathcal{C}$ containing ℓ sinks. Assume further that π has k top-shared clauses. Then π has size at least $2\ell + k - 2$.*

Proof. Consider the sinks that are descendants of $(\bar{a} \vee b)$; say there are ℓ_1 of them (and $\ell_2 = \ell - \ell_1$ remaining sinks). Let S be the set of top-shared clauses. All of the descendant sinks of S are among these ℓ_1 sinks. The subgraph induced by all clauses in π reachable from $(\bar{a} \vee b)$ constitutes a Resolution derivation of $(\bar{a} \vee b)$ from ℓ_1 sinks. Therefore, by Proposition 2, this subgraph must contain at least $2\ell_1 - 1$ clauses. Now consider the subgraph induced by all clauses reachable from $(\bar{c} \vee d)$ where we exclude any shared clause that is not a top-shared clause. This constitutes a Resolution derivaton of $(\bar{c} \vee d)$ from $\ell_2 + k$ sinks (modulo removal of edges between top-shared clauses). Therefore, it must contain at least $2(\ell_2 + k) - 1$ clauses. These two derivations (of $(\bar{a} \vee b)$ and $(\bar{c} \vee d)$) are clause disjoint except for the k clauses in S. Therefore, the entire joint derivation contains at least $(2\ell_1 - 1) + (2(\ell_2 + k) - 1) - k = 2\ell + k - 2$ clauses. □

Let $P_1 \in \mathcal{P}_{ab}$ and $P_2 \in \mathcal{P}_{cd}$ and assume that the sum of the lengths of these paths is L. Let $t_1, ..., t_k$ be shared segments (primal or dual) of P_1 and P_2. Define $JointDerive(P_1, P_2, t_1, ..., t_k)$ to be the following joint derivation of $\bar{a} \vee b$ (or possibly just (b)) and $\bar{c} \vee d$ (or possibly just (d)) from $\mathcal{C}$: for each i, construct $IR(t_i)$; assume this is a derivation of the clause $\bar{x}_i \vee y_i$. Assume that removing

the t_i segments from P_1 yields h_1 intermediate nonempty segments $\{r_j\}_{j=1}^{h_1}$. Likewise, there are h_2 intermediate nonempty segments $\{s_j\}_{j=1}^{h_2}$ in P_2. Derive each r_j and s_j using $IR(r_j)$ and $IR(s_j)$. These $k + h_1 + h_2$ derivations have combined size $2(L - \sum_{i=1}^{k} length(t_i)) - (k + h_1 + h_2)$. Now use the results of the t_i and r_j derivations to derive $(\bar{a} \vee b)$ (or (b)) in an input fashion by adding $k + h_1 - 1$ new clauses. Likewise, derive $(\bar{c} \vee d)$ (or just (d)) by adding $k + h_2 - 1$ new clauses. In total, we have $2(L - \sum_{i=1}^{k} length(t_i)) + k - 2$ clauses.

Let $P_1 \in \mathcal{P}_{ab}$ and $P_2 \in \mathcal{P}_{cd}$ be singular. Let $t_1, ..., t_k$ be the maximal shared segments (primal or dual) of P_1 and P_2. Define the *canonical joint derivation* $CJD(P_1, P_2)$ to be $JointDerive(P_1, P_2, t_1, ..., t_k)$.

Lemma 2. *Let $P_1 \in \mathcal{P}_{ab}$ and $P_2 \in \mathcal{P}_{cd}$ be singular paths and assume that if a and b have distinct underlying variables, then a's variable precedes b's in the ordering (likewise for c and d). $CJD(P_1, P_2)$ is a joint derivation of clauses C_1 and C_2, where C_1 is either $(\bar{a} \vee b)$ or just (b), and C_2 is either $(\bar{c} \vee d)$ or just (d). Moreover, $CJD(P_1, P_2)$ has minimum size over all joint derivations π of C_1' and C_2' where $ResPath(\pi, C_1') = P_1$ and $ResPath(\pi, C_2') = P_2$.*

Proof. Consider any joint derivation π of C_1' and C_2'. Let ℓ be the number of distinct axioms underlying P_1 and P_2 and let k be the number of maximal shared segments. π must have at least ℓ sinks. If π has at least k top-shared clauses it cannot have size smaller than $CJD(P_1, P_2)$ by Lemma 1. Now assume it has $k' < k$ top-shared clauses. Each top-shared clause corresponds to a shared segment of P_1 and P_2. The other shared clauses correspond to subsegments of these shared segments. Therefore, there must be $k - k'$ maximal shared segments such that no subsegment is represented by a shared clause in π. Each such maximal shared segment contains at least one axiom which is not shared in π. Therefore, the number of sinks in π is at least $\ell + k - k'$, so π must have size at least $2(\ell + k - k') + k' - 2 = 2\ell + k - 2 + (k - k') > 2\ell + k - 2$. □

We now show the crucial fact that the paths underlying a minimum Resolution refutation are, without loss of generality, singular. The proof goes by a fairly intense case analysis, which we only sketch here. We do, however, offer some intuition. In [3], we show that, for any derivable, nonempty clause C, there is a smallest derivation of C that is $IR(P)$ for some singular path P in $G_{\mathcal{C}}$. In other words, multiple use of clauses, even axioms, is not helpful. A Resolution refutation is essentially a joint derivation of (x) and $(\bar{x})$ for some variable x. As also shown in [3], independent minimum derivations of (x) and $(\bar{x})$ are sometimes almost twice as large as the minimum joint derivation of the two, so the sharing of clauses between the two derivations can be crucial. Here we simply rule out any benefit of sharing a clause within one side (e.g. the portion used to derive (x)) of the joint derivation.

Lemma 3. *Assume there is a Resolution refutation, π, of $\mathcal{C}$ of size s. Then there is a Resolution refutation of $\mathcal{C}$, π', of size $\leq s$, such that both paths in $ResPath(\pi')$ are singular.*

Proof (sketch). Assume π ends by resolving x and $\bar{x}$. If either $ResPath(\pi, (x))$ or $ResPath(\pi, (\bar{x}))$ is not singular, then there is a clause C in π such that there are at least two paths from x to C or from $\bar{x}$ to C, respectively. Call such a clause *repeated*. If there are k distinct paths from x to C, we say that C is repeated k times with respect to x, or that C has k occurrences with respect to x.

Let C be a repeated clause in π that has no repeated ancestor (if there are no repeated clauses, we are done). We will show how to locally transform π so that we eliminate one occurrence of C and do not add occurrences of any other clause.

Assume without loss of generality that C is repeated with respect to x. Let D be an ancestor of C in π such that there are exactly two distinct paths from D to C and such that no descendant of D has two distinct paths to C. Let r_1 and r_2 denote the two paths from D to C. C must have two distinct literals, say, $(\bar{c} \vee d)$. It may be the case that there is one clause, C_2, on r_2 such that there is one path from $\bar{x}$ to C_2 that is edge-disjoint from r_2 (likewise for C_1 and r_1). There cannot be more than one such clause or one such path by the way we chose C. We will generally assume that C_1 and C_2 exist since the proof is simpler if they don't. Therfore, let D' be a clause reachable from $\bar{x}$ such that there is a path from D' to C_1 (call it r_3) and a node-disjoint path from D' to C_2 (call it r_4). Let r_{31} be r_3 concatenated with the suffix of r_1 from C_1 to C, and let r_{42} be r_4 concatenated with the suffix of r_2 from C_2 to C. So r_{31} and r_{42} are the two distinct paths from D' to C. This entire setup is illustrated in figure 2. We will assume for simplicity that both D and D' contain two distinct literals; the proof is similar if they don't.

Let $Q = ResPath(\pi, C)$, $P = ResPath(\pi, D)$ and $P' = ResPath(\pi, D')$. There are several cases based on how C occurs in P and P'. For instance, the r_1 occurrence of C corresponds to a segment of P that is either Q or $dual(Q)$. Also, the r_1 occurrence of C could either precede or succeed the r_2 occurrence in P. We illustrate one case: assume that the r_2 occurrence of C succeeds the r_1 occurrence in P and that both are Q. Assume that the r_{42} occurrence of C succeeds the r_{31} occurrence in P' and that the r_{42} occurrence is Q while the r_{31} occurrence is $dual(Q)$ (see figure 3).

As described above (after Definition 4), each resolution along, say, path r_2 from C to D corresponds to an extension of (an extension of) the r_2 occurrence of Q. Call a clause in π a neighbor of r_2 if it is a child of any clause in r_2 (except C), but is not in r_2 itself. Let $B^2_1, B^2_2, ..., B^2_{b_2}$ be the neighbors of r_2 that correspond to extending the r_2 occurrence of Q towards the beginning of P and let $E^2_1, ..., E^2_{e_2}$ be the neighbors of r_2 that correspond to extending the r_2 occurrence of Q towards the end of P. Likewise for r_1 and $B^1_1, ..., B^1_{b_1}$ and $E^1_1, ..., E^1_{e_1}$, respectively. Let $B^3_1, ..., B^3_{b_3}$ and $E^3_1, ..., E^3_{e_3}$ be the neighbors of r_3 that extend the r_{31} occurrence of $dual(Q)$ towards the beginning and end of P', respectively (likewise for $B^4_1, ..., B^4_{b_4}$, $E^4_1, ..., E^4_{e_4}$ and the r_{41} occurrence of Q).

Let $IR(B^1)$ be the input derivation that proceeds by resolving $B^1_1, ..., B^1_{b_1}$ in order and let B^1 denote the final clause in this derivation (likewise for all the B^i's and E^i's). It must be the case that D is the result of resolving C with B^1 and E^2

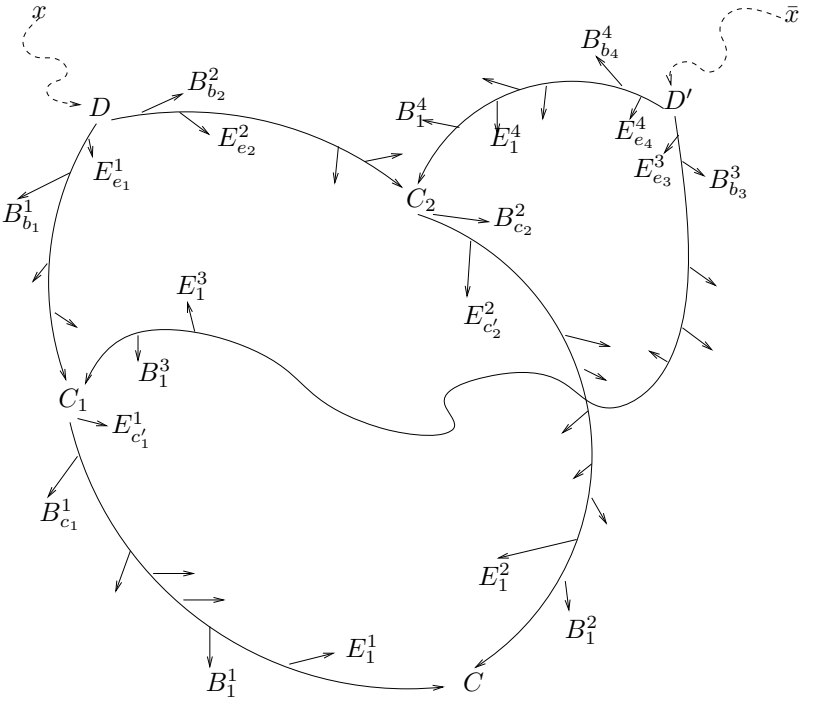


Fig. 2. Original derivation

(see figure 4). To derive D' (or, in fact, something stronger), let $B^1(c_1)$ denote the $(c_1 - 1)$th derived clause in B^1 and let E^{31} denote the result of resolving $B^1(c_1)$ with E^3. Likewise, let B^{42} denote the result of resolving $B^2(c_2)$ with B^4. Finally, let E^{42} denote the result of resolving $E^2(c'_2)$ with E^4. The clause that results from resolving C with E^{31}, B^{42} and E^{42} successively must be a subclause of D'. Now we must compare the size of the modified derivation with the size of the original derivation. In the original, each B^i_j and E^i_j clause gives rise to a new derived clause, so there are $K = \sum_{i=1}^{4} b_i + e_i$ derived clauses along the paths $r_1, ..., r_4$. In the modified derivation, the total number of derived clauses in the input derivations B^i and E^i is $\sum_{i=1}^{4}(b_i - 1) + (e_i - 1) = K - 8$. To finish deriving D, we create one intermediate derived clause; to finish D', we create five. Therefore the modified derivation is no bigger. □

Now we show that we can assume the pair of singular paths underlying a minimum Resolution refutation obeys special properties. In light of Lemma 2, we call a pair of singular, end-contradictory paths P_1, P_2 *minimum* if they minimize the expression $f(P_1, P_2) \equiv 2\ell(P_1, P_2) + k(P_1, P_2) - 1$. In other words, they generate a minimum size refutation (the -1 term in the expression replaces the -2 in the size of $CJD(P_1, P_2)$ because we count the empty clause).

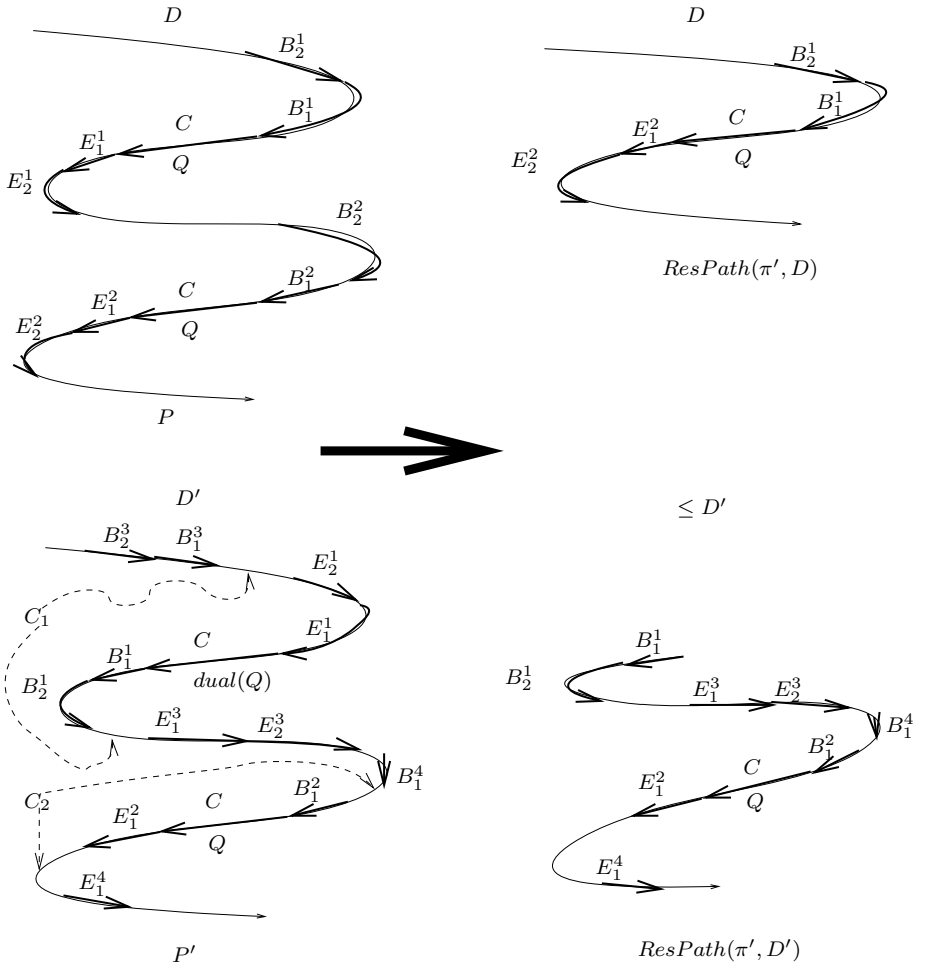


Fig. 3. Transformation of paths

Consider the following properties of two singular paths P_1 and P_2.
Property I: Let $s_1 \prec_{P_1} \cdots \prec_{P_1} s_k$ be the maximal primal shared segments of P_1 and P_2. Then $s_k \prec_{P_2} \cdots \prec_{P_2} s_1$.
Property II: Let $t_1 \prec_{P_1} \cdots \prec_{P_1} t_\ell$ be the maximal dual shared segments of P_1 with respect to P_2. Then $dual(t_1) \prec_{P_2} \cdots \prec_{P_2} dual(t_\ell)$.
Property III: Let $s_1 \prec_{P_1} \cdots \prec_{P_1} s_k$ be the maximal primal shared segments of P_1 and P_2 and let $t_1 \prec_{P_1} \cdots \prec_{P_1} t_\ell$ be the maximal dual shared segments of P_1 with respect to P_2. For any i, j, $t_i \prec_{P_1} s_j$ if and only if $dual(t_i) \prec_{P_2} s_j$.
Property IV: All shared segments of P_1 and P_2 occur in $core(P_1)$ and $core(P_2)$.

Lemma 4. *Every minimum pair of singular, end-contradictory paths must satisfy Properties I-IV.*

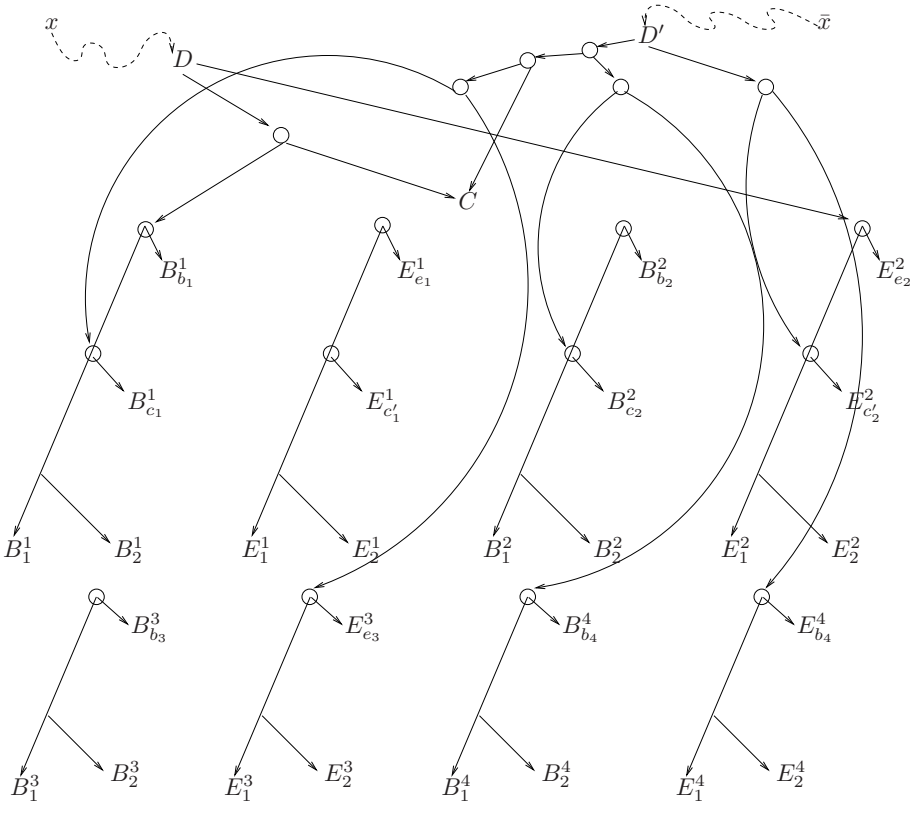


Fig. 4. Modified derivation

Proof. Our general strategy is to take a pair of singular, end-contradictory paths P_1, P_2 that violate one of the properties and transform them into a pair of singular, end-contradictory paths P_1', P_2' such that $f(P_1', P_2') < f(P_1, P_2)$.

Consider Property I. If P_1 and P_2 violate the property, then there is some $i < j$ such that $s_i \prec_{P_2} s_j$. Let P_1' be the segment of P_1 starting at the beginning of s_i and ending at the end of s_j. Likewise, let P_2' be the segment of P_2 that starts at the beginning of s_i and ends at the end of s_j. Assume, without loss of generality, that $\mathit{length}(P_1') \leq \mathit{length}(P_2')$. Let P_2'' be the path P_2 with P_2' replaced by P_1'. It must be the case that P_2'' is singular since otherwise there would have been a shared segment in between s_i and s_j in P_2. Futhermore, P_1 and P_2'' are clearly end-contradictory. Finally, $f(P_1, P_2'') < f(P_1, P_2)$ since both the number of underlying clauses and the number of maximal shared segments have gone down. Property II follows in the same way by looking at P_1 and $\mathit{dual}(P_2)$.

Consider Property IV. Let P_1, P_2 be singular, end-contradictory paths. Assume, without loss of generality, that $\mathit{core}(P_1) \neq P_1$. Let $\mathit{core}(P_1)$ go from a to $\bar{a}$ and let P_1 end at x. Assume P_2 starts at b and ends at $\bar{x}$ (b may equal x). Let

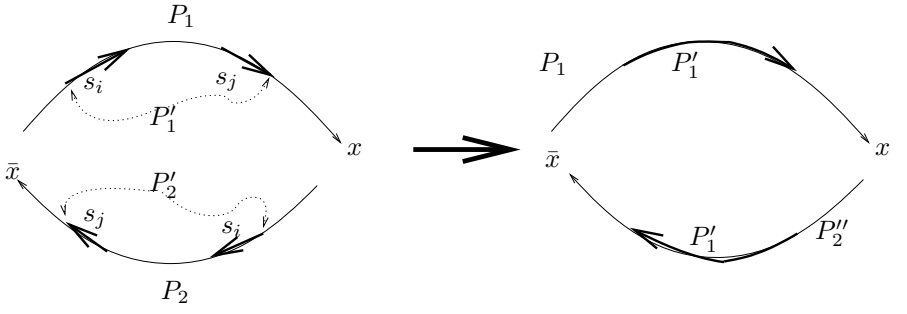


Fig. 5. Forcing Property I

s be the maximal shared segment that ends as late as possible in P_1. Assume s goes from c to d such that d occurs after $\bar{a}$ in P_1. If s is a primal shared segment, then let P_1' be the segment of P_1 that goes from a to d. Let Q_1 be the segment of P_1 that goes from d to x and let Q_2 be the segment of P_2 that goes from d to $\bar{x}$. Let $P_2' = Q_2 \circ dual(Q_1)$. Note that P_1' and P_2' are end-contradictory and singular. Also, $f(P_1', P_2') < f(P_1, P_2)$ since the number of shared segments has gone down. If s is a dual shared segment, then again let P_1' be the segment of P_1 that goes from a to d. If $\bar{d}$ occurs at or after $\bar{b}$ in P_2, then let P_2' be the segment of P_2 that goes from b to $\bar{d}$. Otherwise, let Q_1 be the segment of P_1 from d to x. Let Q_2 be the (possibly empty) segment of P_2 from $\bar{b}$ to $\bar{x}$, and let Q_3 be the segment of P_2 from b to $\bar{d}$. Set $P_2' = Q_1 \circ dual(Q_2) \circ Q_3$. Again, P_1' and P_2' are singular and end-contradictory and $f(P_1', P_2') < f(P_1, P_2)$.

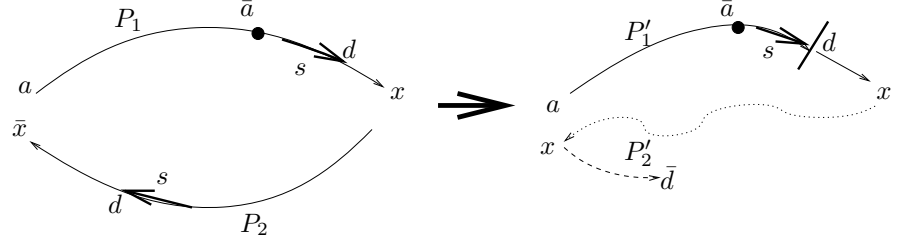


Fig. 6. Forcing Property IV

Finally, consider Property III. We assume that Properties I, II and IV hold. If P_1 and P_2 violate the property, then there is a primal shared segment s and a dual shared segment t such that, without loss of generality, $t \prec_{P_1} s$, but $s \prec_{P_2} dual(t)$, and furthermore there are no shared segments between t and s in P_1. Let c, d be the endpoints of t, and g, h the endpoints of s. Let Q_1 be the segment of P_1 from d to h, and let Q_2 be the segment of P_2 from h to $\bar{c}$. Let $P_1' = Q_1 \circ Q_2$. Note that P_1' is singular. Let Q_3 be the segment of P_1 from the end of $core(P_1)$ to the end of P_1 (say P_1 ends at x). Let Q_4 be the segment of P_2 from $\bar{c}$ to the end and let Q_5 be the segment of P_1 from the beginning to c. Let

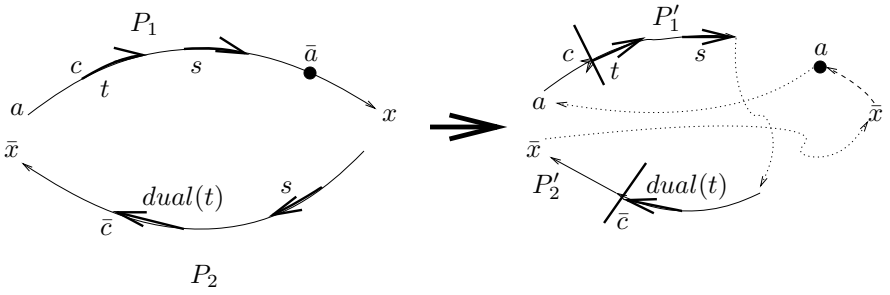


Fig. 7. Forcing Property III

$P_2' = Q_4 \circ dual(Q_3) \circ Q_5$. P_2' may not be singular, so let $P_2'' = sing(P_2')$. Clearly P_1' and P_2'' are end-contradictory and singular. Also, $f(P_1', P_2'') < f(P_1, P_2)$. □

4 The Algorithm

Our algorithm for finding a minimum Resolution refutation will use dynamic programming in a similar way that, say, the Bellman-Ford algorithm does. It would be sufficient to find a minimum pair of singular, end-contradictory paths P_1, P_2, but it is unclear how to limit our search to singular paths, since arbitrary extensions of singular paths are not necessarily singular. On the other hand, if we have two non-singular, end-contradictory paths, there does not seem to be a simple characterization of the size of a smallest Resolution refutation in terms of the lengths of the paths and the lengths of any shared segments. We get around this problem by defining a generalized cost of two arbitrary paths such that the cost is at least the size of the minimum joint derivation based on the paths, but is equal to this size in the case where both paths are singular. Therefore, optimizing over all pairs of end-contradictory paths with respect to this generalized cost must find a minimum since we know that the minimum is achieved by a pair of singular paths.

Another ingredient to the algorithm is that we can focus on pairs of paths that obey properties I-IV (we will explain what this means for non-singular paths shortly). In particular, the structure provided by these properties allows us to do dynamic programming where the recursion is on the number of shared segments between a pair of paths. The recursion is based on the following idea. The reason a pair of paths P_1 and P_2 that minimize the cost function may not each be of minimum length is that, while longer, they benefit by sharing more clauses. If we demand that P_1 and P_2 have a shared segment with specified endpoints, however, then that segment should be as short as possible; likewise, for any segment of, say, P_1 with specified endpoints that is guaranteed not to overlap any shared segment. By doing this, we isolate segments of P_1 and P_2 that we can locally optimize and then concentrate on the remainder of the paths.

For two paths P_1 and P_2, define $cost(P_1, P_2, k)$ to be the minimum of the expression

$$length(P_1) + length(P_2) - \sum_{i=1}^{r} length(s_i) - \sum_{j=1}^{q} length(t_j)$$

over all choices of $s_1, ..., s_r, t_1, ..., t_q$, $r + q = k$, such that $s_1 \prec_{P_1} \cdots \prec_{P_1} s_r$ are (possibly empty) primal shared segments of P_1 and P_2, $t_1 \prec_{P_1} \cdots \prec_{P_2} t_q$ are (possibly empty) dual shared segments of P_1 with respect to P_2, all of the s_i's and t_j's are edge-disjoint from one another and they obey Properties I-III. Given four literals a, b, c, d and a natural number k, define $cost(a, b, c, d, k)$ to be the minimum over all paths $P_1 \in \mathcal{P}_{ab}$ and $P_2 \in \mathcal{P}_{cd}$ of $cost(P_1, P_2, k)$.

The algorithm will compute $cost(a, b, c, d, k)$ for all literals a, b, c, d and all $0 \leq k \leq m$, and will store with each entry a pair of paths and set of shared segments that achieve that cost. To find a minimum Resolution refutation, we search for litarals a, b, x and a number k that minimize

$$2(cost(a, \bar{a}, b, \bar{b}, k) + cost(\bar{a}, x, \bar{b}, \bar{x}, 0)) + k - 1.$$

The reason for the two *cost* terms is Property IV, which assures us that we need not consider any shared segments outside of the cores of the paths. For fixed k, let P_1, P_2 be the pair of paths that minimize the first term in this expression and let $s_1, ..., s_r, t_1, ..., t_q$ be the shared segments. Let P_1', P_2' be the paths that minimize the second term in this expression. Let $Q_1 = P_1 \circ P_1'$ and let $Q_2 = P_2 \circ P_2'$. Then $JointDerive(Q_1, Q_2, s_1, ..., s_r, t_1, ..., t_q)$ is minimum for this value of k. We then simply optimize over all values of k.

To begin, for all literals a, b, set $B[a, b]$ to the length of a shortest path in $\mathcal{P}_{ab}$. This can be done using Bellman-Ford, for example. For all literals a, b, c, d, set $cost(a, b, c, d, 0)$ to $B[a, b] + B[c, d]$. To compute a general entry in $cost()$ where k is nonzero, let P_1 and P_2 be the paths that achieve the minimum corresponding

For all literals a, b
  $B[a, b] \longleftarrow \min\{length(P) \mid P \in \mathcal{P}_{ab}\}$
For all literals a, b, c, d
  $cost(a, b, c, d, 0) \longleftarrow B[a, b] + B[c, d]$
For $k = 1$ to m do
  For all literals a, b, c, d
    For all literals x, y
      $tmp \longleftarrow \min\{B[a, x] + B[y, d] + B[x, y] + cost(y, b, c, x, k - 1),$
        $B[a, x] + B[b, \bar{y}] + B[x, y] + cost(y, b, \bar{x}, d, k - 1),$
        $B[y, b] + B[\bar{x}, d] + B[x, y] + cost(a, x, c, \bar{y}, k - 1)\}$
      If $tmp < cost(a, b, c, d, k)$ then $cost(a, b, c, d, k) \longleftarrow tmp$

Output $\min_{0 \leq k \leq m} \min_{a,b,x} 2(cost(a, \bar{a}, b, \bar{b}, k) + cost(\bar{a}, x, \bar{b}, \bar{x}, 0)) + k - 1$

Fig. 8. Computing the size

to the entry in question. By Properties I-III, there are three cases. **(1)** There are no dual shared segments of P_1 with respect to P_2. Therefore, the first shared segment in P_1 (in order of appearance) is a primal shared segment s_1 that is the last shared segment in P_2. **(2)** The first shared segment in P_1 is a dual shared segment t_1 and $dual(t_1)$ is the first shared segment in P_2. **(3)** The last shared segment in P_1 is a dual shared segment t_q and $dual(t_q)$ is the last shared segment in P_2.

Therefore, to compute $cost(a, b, c, d, k)$, we take the minimum over all literals x, y of the minimum of **(1)** $B[a, x] + B[y, d] + B[x, y] + cost(y, b, c, x, k - 1)$; **(2)** $B[a, x] + B[b, \bar{y}] + B[x, y] + cost(y, b, \bar{x}, d, k - 1)$; **(3)** $B[y, b] + B[\bar{x}, d] + B[x, y] + cost(a, x, c, \bar{y}, k - 1)$. The algorithm for computing the size of a smallest Resolution refutation is summarized in figure 8. It is not hard to see that it runs in time $O(n^6 m)$. As mentioned above, one can produce a minimum refutation by keeping track of the paths and shared segments that achieve the minima. This adds nothing to the asymptotic complexity.

References

1. M. Alekhnovich, S. Buss, S. Moran, and T. Pitassi. Minimum propositional proof length is NP-hard to linearly approximate. *JSL: Journal of Symbolic Logic*, 66, 2001.
2. Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, March 1979.
3. J. Buresh-Oppenheim and D. Mitchell. Minimum witnesses for unsatisfiable 2CNFs. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2006.
4. S.A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158, New York, 1971. Association for Computing Machinery.
5. Alvaro del Val. On 2-SAT and Renamable Horn. In *AAAI'2000, Proc. 17th (U.S.) National Conference on Artificial Intelligence*. AAAI Press/The MIT Press, 2000.
6. S. Even, A. Itai, and A Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4), 1976.

Polynomial Time SAT Decision for Complementation-Invariant Clause-Sets, and Sign-non-Singular Matrices

Oliver Kullmann*

Computer Science Department, University of Swansea
Swansea, SA2 8PP, UK
O.Kullmann@Swansea.ac.uk
http://cs-svr1.swan.ac.uk/ csoliver

Abstract. We study *complement-invariant clause-sets* F, where for every clause $C \in F$ we have $\overline{C} = \{\overline{x} : x \in C\} \in F$, i.e., F is closed under elementwise complementation of clauses. The *reduced deficiency* of a clause-set F is defined as $\delta_{\rm r}(F) := \frac{1}{2}(\delta(F) - n(F))$, where $\delta(F) = c(F) - n(F)$ is the difference of the number of clauses and the number of variables, while the *maximal reduced deficiency* is $\delta^*_{\rm r}(F) := \max_{F' \subseteq F} \delta_{\rm r}(F') \geq 0$. We show polynomial time SAT decision for complement-invariant clause-sets F with $\delta^*_{\rm r}(F) = 0$, exploiting the (non-trivial) decision algorithm for sign-non-singular (SNS) matrices given by [Robertson, Seymour, Thomas 1999; McCuaig 2004]. As an application, hypergraph 2-colourability decision is considered. Minimally unsatisfiable complement-invariant clause-sets F fulfil $\delta_{\rm r}(F) = \delta^*_{\rm r}(F)$, and thus we immediately obtain polynomial time decidability of minimally unsatisfiable complement-invariant clause-sets F with $\delta_{\rm r}(F) = 0$, but we also give more direct algorithms and characterisations (especially for sub-classes). The theory of autarkies is the basis for all these considerations.

1 Introduction

The *deficiency* $\delta(F) = c(F) - n(F)$ of clause-sets (where $c(F)$ is the number of clauses, and $n(F)$ is the number of variables) is an interesting parameter, allowing polynomial time SAT decision for clause-sets F with bounded maximal deficiency (maximised over all sub-clause-sets; see [3,4]). We make the first step towards an analogous poly-time hierarchy, where we consider *only complement-invariant* clause-sets F (for every $C \in F$ also $\overline{C} \in F$ holds), while we strengthen the notion of deficiency to *reduced deficiency* $\delta_{\rm r}(F) = \frac{1}{2}(\delta(F) - n(F))$, which is the deficiency of a "core half" of F, containing one representative C from every complementary pair $C, \overline{C}$ of clauses. We prove polynomial time SAT decision for complement-invariant clause-sets with the maximal reduced deficiency equal to its minimal value 0. The proof is based on a recent breakthrough by [13,11],

* Supported by grant EPSRC GR/S58393/01.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 314–327, 2007.

where several long outstanding problems (open for up to 93 years) have been solved. As an application we obtain polynomial time decision of 2-colourability of hypergraphs with the maximal deficiency equal to its minimal value 0 (where the deficiency of an hypergraph is the difference of the number of hyperedges and the number of variables).

We exploit heavily the relationship between autarky theory and matrix analysis. More specifically, we exploit *qualitative matrix analysis* (QMA), matrix analysis modulo the equivalence relation between matrices given by having the same sign pattern. [2] describes the foundations of QMA in qualitative economics:

> Qualitative economics is usually considered to have originated with the work of Samuelson who discussed the possibility of determining unambiguously the qualitative behavior of solution values of a system of equations. In his pioneering paper Lancaster put it this way:
> Economists believed for a very long time, and most economists would still hope it to be so, that a considerable body of sensible economic propositions could be expressed in a qualitative way, that is, in a form in which the algebraic sign of some effect is predicted from a knowledge of the signs, only, of the relevant structural parameters of the system.

For example, that a system of (differential) equations has a unique solution is (often) controlled by a square matrix A having non-zero determinant. Now in qualitative matrix analysis we want $\det(A) \neq 0$ independent of the magnitude of the entries of A, if only the signs are preserved. This example leads to two fundamental notions of QMA, which are also central for this article: A square matrix A is called an **SNS-matrix** if all matrices A' with the same sign pattern are non-singular, while more generally a matrix A is called an **L-matrix** if all matrices A' with the same sign pattern have linearly independent rows. As already remarked in [7], L-matrices correspond 1-1 to complement invariant clause-sets which are lean (have no non-trivial autarky); the main results of this article are related to SNS-matrices, which in turn correspond 1-1 to such lean complement invariant clause-sets which have reduced deficiency 0.

Likely the best way to determine whether an arbitrary matrix is an L-matrix is to determine whether the corresponding complement invariant clause-set is lean (for example using the method as discussed in [9]). However for SNS-matrices finally a polynomial time algorithm was found in [13], which is the basis for deciding leanness and minimally unsatisfiability for complement invariant clause-sets with reduced deficiency 0 in polynomial time (Corollary 14). In Theorem 20 these result are strengthened by showing that actually satisfiability is decidable in polynomial time for complement invariant clause-sets with *maximal* reduced deficiency 0. As an immediate application, in Corollary 22 we obtain, that 2-colourability for hypergraphs G with maximal deficiency 0 is decidable in polynomial time.

2 Some General Theory of Autarky Systems

We use standard (boolean) clause-sets F, which are finite sets of clauses here, where a clause is a finite set of non-clashing literals; the empty clause is $\bot$, the empty clause-set $\top$. Application of partial assignments φ to F is denoted by $\varphi * F$ (substituting truth values for the literals touched by φ with subsequent simplification), while by $V * F$ for some set V of variables the operation of crossing out the variables of V from F is denoted (that is, $V * F = \{\{x \in C : \mathrm{var}(x) \notin V\} : C \in F\}$). Furthermore $F[V] := ((\mathrm{var}(F) \setminus V) * F) \setminus \{\bot\}$ is the restriction of F to the variable-set V.

A partial assignment φ is an **autarky** for a clause-set F if every clause $C \in F$ touched by φ (that is, $\mathrm{var}(\varphi) \cap \mathrm{var}(C) \neq \emptyset$) is actually satisfied by φ; the systematic exploration of autarkies started with [5], where the reader may find more background information. The main algorithmic use of autarkies φ for F is *autarky reduction*, the transition from F to the satisfiability-equivalent $\varphi * F$ (which is the sub-clause-set of F given by all clauses not touched by φ). An autarky φ for F is called **balanced** if also $\overline{\varphi}$ is an autarky for F, where $\overline{\varphi}$ is the pointwise complement of φ. Balanced autarkies share many general properties with ordinary autarkies. Since there are many more types of "special autarkies", a general theory is needed, provided by the theory of **autarky systems**, which axiomatically specifies the properties needed of the map $F \mapsto \mathcal{A}(F) \subseteq \mathrm{Auk}(F)$ from clause-sets F to sub-monoids $\mathcal{A}(F)$ of the autarky monoid so that the special autarkies behave like ordinary autarkies. See [6,7,8] for precise definitions and some fundamental results. All autarky systems we have encountered fulfil, besides the basic requirements cast in the notion of an autarky system, the following five fundamental properties (or can be easily extended to fulfil them), which are collected in the notion of a **normal autarky system** (see [8] for the most current version of this notion):

1. $\mathcal{A}$ is **iterative**, if $\mathcal{A}$-autarkies of sub-clause-sets of F, obtained by $\mathcal{A}$-autarky reduction, are still in $\mathcal{A}(F)$.
2. $\mathcal{A}$ is called **standardised**, if $\mathcal{A}$-autarkies can be set arbitrarily on variables not in F.
3. $\mathcal{A}$ is **$\bot$-invariant**, if $\mathcal{A}(F)$ is invariant against addition or deletion of the empty clause $\bot$.
4. $\mathcal{A}$ is **stable under variable elimination**, if for every set V of variables the $\mathcal{A}$-autarkies of $V * F$ which do not use V are exactly the $\mathcal{A}$-autarkies of F which do not use V.
5. $\mathcal{A}$ is **invariant under renaming**, if renaming of clause-sets F carries over to $\mathcal{A}(F)$.

In this article we consider four normal autarky systems:

- Auk, the full autarky system (all autarkies)
- BAuk, the system of balanced autarkies (see Section 3)

– LAuk resp. BLAuk, the normalised system created from simple linear autarkies resp. from simple balanced linear autarkies (see Section 4).[1]

Let $\mathcal{A}$ be a normal autarky system. A clause-set F is called **$\mathcal{A}$-lean** if F has no non-trivial $\mathcal{A}$-autarky (one which touches F); there is a largest $\mathcal{A}$-lean sub-clause-set of F, called the **$\mathcal{A}$-lean kernel**. F is called **$\mathcal{A}$-satisfiable** if the $\mathcal{A}$-lean kernel is $\top$ (which is equivalent to the existence of an $\mathcal{A}$-autarky for F which actually satisfies F), while otherwise F is called **$\mathcal{A}$-unsatisfiable**. $\top$ is $\mathcal{A}$-lean, and if $F \neq \top$ is $\mathcal{A}$-lean, then F is $\mathcal{A}$-unsatisfiable; F is $\mathcal{A}$-lean iff $F \setminus \{\bot\}$ is $\mathcal{A}$-lean iff $F \cup \{\bot\}$ is $\mathcal{A}$-lean. If $\mathcal{A}$ is the full autarky system, then we just speak of satisfiability, unsatisfiability, and leanness.

Definition 1. *A clause-set F is called **minimally $\mathcal{A}$-unsatisfiable**, if F is $\mathcal{A}$-unsatisfiable, while every $F' \subset F$ is $\mathcal{A}$-satisfiable.*

1. For $\mathcal{A}$ as the full autarky system we obtain ordinary minimal unsatisfiability.
2. $F \neq \top$ is minimally $\mathcal{A}$-lean (that is, F is $\mathcal{A}$-lean, while every $F' \subset F$ is not $\mathcal{A}$-lean) if and only if F is minimally $\mathcal{A}$-unsatisfiable.

Definition 2. *A clause-set F is called **barely $\mathcal{A}$-lean** (in generalisation of the notion of a "barely L-matrix" in [2]) if F is $\mathcal{A}$-lean, while for every clause $C \in F$ the clause-set $F \setminus \{C\}$ is not $\mathcal{A}$-lean.*

1. $\top$ is barely $\mathcal{A}$-lean, while if $c(F) = 1$, then F is not barely $\mathcal{A}$-lean.
2. If F is minimally $\mathcal{A}$-unsatisfiable and $c(F) \neq 1$, then F is barely $\mathcal{A}$-lean.
3. If $\mathcal{A}$ is the full autarky system, then we speak of "barely lean".
4. If F is barely $\mathcal{A}$-lean then $\bot \notin F$ (otherwise also $F \setminus \{\bot\}$ would be $\mathcal{A}$-lean).

Lemma 3. *If for a normal autarky system $\mathcal{A}$, deciding whether a non-trivial autarky exists and finding one if existent, is solvable in polynomial time, then the following decision problems are solvable in polynomial time:*

1. *$\mathcal{A}$-satisfiability and $\mathcal{A}$-unsatisfiability;*
2. *minimal $\mathcal{A}$-unsatisfiability;*
3. *A-leanness;*
4. *barely $\mathcal{A}$-leanness.*

[1] The problem with simple (balanced) linear autarkies, which makes "normalisation" necessary, is that these autarky systems are not "iterative", that is, in general it is not the case that if φ is a simple (balanced) linear autarky for F and if ψ is a simple (balanced) linear autarky for $\varphi * F$, then $\psi \circ \varphi$ is a (balanced) linear autarky for F, because ψ might invalidate the (balanced) simple-linear-autarky-condition for clauses which had been already satisfied by φ (and thus were removed). "Normalisation" just adds these compositions of "iterated autarkies" to the autarky monoid, and for the resulting (balanced) linear autarkies the adjective "simple" is dropped; see Subsections 4.4 and 4.6 in [5] for the special case of linear autarkies, and Lemma 8.4 in [7] for the general case.

Definition 4. *A clause-set F is a* ***generalised sum*** *of clause-sets F_1, F_2 if $var(F_1) \cap \mathrm{var}(F_2) = \emptyset$ with $\mathrm{var}(F_1), \mathrm{var}(F_2) \neq \emptyset$, and there is F_2' with $F = F_1 \cup F_2'$ (disjoint union), such that F_2' is obtained from F_2 by adding literals over $\mathrm{var}(F_1)$ to clauses from F_2 (that is, there exists a bijection $\alpha : F_2 \to F_2'$ such that for all $C \in F_2$ we have $\alpha(C) \supseteq C$ and $\mathrm{var}(\alpha(C) \setminus C) \subseteq \mathrm{var}(F_1)$).*

1. Let F be a generalised sum of F_1, F_2:
   (a) $F = F_1 \cup F_2'$ and $\mathrm{var}(F) = \mathrm{var}(F_1) \cup \mathrm{var}(F_2)$, and thus $\delta(F) = c(F) - n(F) = (c(F_1) + c(F_2)) - (n(F_1) + n(F_2)) = \delta(F_1) + \delta(F_2)$.
   (b) $F_2 \setminus \{\bot\} = F[\mathrm{var}(F_2)]$.
   (c) If F is $\mathcal{A}$-lean, then so is F_2.
   (d) If F_1, F_2 are $\mathcal{A}$-lean, then so is F.
2. If F is any clause-set and $F_1 \subset F$ with $\emptyset \subset \mathrm{var}(F_1) \subset \mathrm{var}(F)$ and such that for $C \in F \setminus F_1$ we have $\mathrm{var}(C) \not\subseteq \mathrm{var}(F_1)$, then F is a generalised sum of F_1, F_2 for $F_2 := F[\mathrm{var}(F) \setminus \mathrm{var}(F_1)]$.

Definition 5. *A clause-set F is* ***$\mathcal{A}$-indecomposable*** *(in generalisation of the notion of an "L-indecomposable matrix" in [2]) if F is not the generalised sum of $\mathcal{A}$-lean clause-sets F_1, F_2; otherwise F is called* ***$\mathcal{A}$-decomposable****.*

1. Note that if F is $\mathcal{A}$-decomposable, then F is necessarily $\mathcal{A}$-lean, while if F is not $\mathcal{A}$-lean, then F is $\mathcal{A}$-indecomposable.
2. If $\mathcal{A}$ is the full autarky system, then we speak of "autarky indecomposable".
3. F is $\mathcal{A}$-decomposable iff $F \setminus \{\bot\}$ is $\mathcal{A}$-decomposable iff $F \cup \{\bot\}$ is $\mathcal{A}$-decomposable.

Generalising Theorem 2.2.5 in [2]:

Lemma 6. *A clause-set F with $c(F) \geq 2$ is minimally $\mathcal{A}$-unsatisfiable if and only if the following two conditions hold:*
(i) F is barely $\mathcal{A}$-lean
(ii) F is $\mathcal{A}$-indecomposable.

Proof. Clearly the two conditions are necessary; it remains to see that they are sufficient. Since F is barely $\mathcal{A}$-lean, F is $\mathcal{A}$-lean, and thus $\mathcal{A}$-unsatisfiable, and furthermore we have $\bot \notin F$. Now consider $C \in F$, and assume that $F \setminus \{C\}$ is not $\mathcal{A}$-satisfiable. Then there is a non-trivial autarky φ for $F \setminus \{C\}$ such that $F_1 := \varphi * (F \setminus \{C\})$ is $\mathcal{A}$-lean, where $\top \subset F_1 \subset F \setminus \{C\}$. Now, by Remark 2 to Definition 4, F is a generalised sum of F_1 and $F_2 := F[\mathrm{var}(F) \setminus \mathrm{var}(F_1)]$ (clauses from $F \setminus F_1$ different from C contain some literal satisfied by φ, while C contains a literal falsified by φ), contradicting $\mathcal{A}$-indecomposability of F by Remark 1c to Definition 4. □

We conclude this section on general autarky systems by regarding the complexity of the basic decision problems for the full autarky system:

1. satisfiability/unsatisfiability decision is NP/coNP-complete;
2. minimally unsatisfiability decision is D^P-complete ([12]);
3. leanness decision is coNP-complete ([7]);
4. barely lean decision is D^P-complete ([10]);
5. autarky decomposability decision is in Σ_2 (it is not known whether autarky decomposability decision is Σ_2-complete).

3 Balanced Autarkies

[7] introduced **balanced autarkies** for clause-sets F, which are partial assignments φ such that for every clause $C \in F$ touched by φ there exists a satisfied *as well as* a falsified literal in C. The set of all balanced autarkies for F is $\mathbf{BAuk}(\boldsymbol{F})$; it is BAuk a normal autarky system. We use the following phrases:

- "BAuk-satisfiable" resp. "BAuk-unsatisfiable" is called *balanced satisfiable* resp. *balanced unsatisfiable*;
- "minimally BAuk-unsatisfiable" is called *minimally balanced unsatisfiable*;
- "(barely) BAuk-lean" is called *(barely) balanced lean*;
- "BAuk-indecomposable" is called *balanced autarky-indecomposable*.

The complexities of the basic decision problems for balanced autarkies are likely the same as for general autarkies (see the end of Section 2), but proven at this time is only the coNP-completeness of balanced leanness decision (as remarked in [7]). For a partial assignment by $\overline{\boldsymbol{\varphi}}$ we denote the pointwise complement of φ, that is, $\mathrm{var}(\overline{\varphi}) = \mathrm{var}(\varphi)$ and $\overline{\varphi}(v) = \overline{\varphi(v)}$.

Lemma 7. *The following assertions are equivalent for a partial assignment φ and a clause-set F:*

1. *φ is a balanced autarky for F*
2. *φ and $\overline{\varphi}$ are autarkies for F.*

Thus complementation of partial assignments yields an automorphism of the balanced autarky monoid $\mathrm{BAuk}(F)$.

3.1 *L*-Matrices and SNS-Matrices

All matrices in this article have real entries. For a matrix M the *sign pattern* $\mathrm{sgn}(M)$ is the $\{-1, 0, +1\}$-matrix $\mathrm{sgn}(M)$ of the same dimension given by entrywise sgn-formation, while the *null pattern* of M is $\mathrm{sgn}(|M|)$ (a $\{0, 1\}$-matrix), where $|M|$ denotes entrywise absolute-value formation.

For a clause-set F let $M(F)$ be the *clause-variable matrix* of F (see Section 3 in [7] for more details). As shown in Section 5 of [7], a clause-set F is balanced lean if and only if the matrix $M(F)^{\mathrm{t}}$ is an *L-matrix*, where a matrix M is called an L-matrix if each matrix with the same sign pattern as M has linearly independent rows. L matrices have at least as many columns as rows, and thus for balanced lean clause-sets F we have $\delta(F) \geq 0$ (see Lemma 19 for a more general and stronger statement). Square L-matrices are called *SNS-matrices* ("sign-non-singular matrices"), which are characterised by the condition that every square matrix with the same sign pattern is invertible (non-singular); so a clause-set F with $\delta(F) = 0$ is balanced lean iff $M(F)^{\mathrm{t}}$ is an SNS-matrix. We also have the inverse directions (so that the notions of L- and SNS-matrices are fully captured by balanced lean clause-sets):

1. A matrix M without repeated columns is an L-matrix if and only if M has no zero rows and "the" clause-set F with $M(F)^{\mathrm{t}} = M$ is balanced lean.

2. A matrix M is an SNS-matrix if and only if M neither has zero rows nor repeated columns and "the" clause-set F with $M(F)^{\mathrm{t}} = M$ is balanced lean and fulfils $\delta(F) = 0$.

The special treatment of rows and columns is necessary due to the use of clause-*sets*, which contract multiple clauses and also eliminate purely "formal" variables (which do not occur).

For a square $\{-1, 0, +1\}$-matrix A of order $n \in \mathbb{N}_0$ the following conditions are equivalent (see [2] for the (easy) proofs):

1. A is an SNS-matrix;
2. $\det(A) \neq 0$ and all non-null terms in the determinant expansion of A have the same sign (that is, there is $\varepsilon \in \{-1, +1\}$ such for all permutations $\pi \in S_n$ of $\{1, \ldots, n\}$ in case of $\prod_{i=1}^n A_{i,\pi(i)} \neq 0$ we have $\mathrm{sgn}(\pi) \cdot \prod_{i=1}^n A_{i,\pi(i)} = \varepsilon$).
3. $\det(A) \neq 0$ and $\mathrm{per}(|A|) = |\det(A)|$, where $\mathrm{per}(A) = \sum_{\pi \in S_n} \prod_{i=1}^n A_{i,\pi(i)}$ denotes the permanent of a square matrix A.
4. $\det(A) \neq 0$ and for every square matrix M of order n we have $\mathrm{per}(M * |A|) = |\det(M * A)|$, where $M * A$ denotes the pointwise (Hadamard-)product of matrices of the same dimension.

The "Pólya-Problem", as discussed in [13], is the problem to determine whether for a square $\{0, 1\}$-matrix A there exists an SNS-$\{-1, 0, +1\}$-matrix B with the same null pattern. Algorithm 9.7 there decides in polynomial time whether for input A the matrix B exists, and also computes B if it exists.[2] This algorithm yields poly-time decision of the SNS-property for square $\{-1, 0, +1\}$-matrices M as follows (while deciding the (general) L-matrix-property is coNP-complete):

1. If $\det(M) = 0$ then M is not an SNS-matrix.
2. Let $A := |M|$.
3. If A has no associated SNS-matrix B then M is not an SNS-matrix.
4. Otherwise M is an SNS-matrix iff $|\det(B)| = |\det(M)|$.

A problem remains: In this way we can decide whether M is an SNS-matrix, but in case M is not an SNS-matrix, how do we find (in polynomial time) a matrix M' with the same sign pattern as M which is singular ? The critical step is Step 3 in the above procedure, where one has to examine the different obstructions studied in [13]. It seems plausible, that from these obstructions one can compute a witness M', but on the other hand it doesn't seem to be straightforward, and so we formulate the remaining algorithmic problem as a conjecture:

[2] [13] uses an equivalent formulation based on the notion of "Pfaffian orientation". For an oriented graph D (undirected graphs, where additionally every edge has an orientation), in the skew-symmetric $\{0, \pm 1\}$-matrix $A'(D)$ of order $|V(D)|$ let entries ± 1 denote the direction of edges. A *Pfaffian orientation* of an undirected graph G is an orientation σ of the edges of G, yielding an oriented graph G_σ, such that the square root of the determinant of matrix $A'(G_\sigma)$ is the number of perfect matchings of G. When applied to bipartite G, the problem of deciding whether a Pfaffian orientation of G exists (and then finding one) is equivalent to the Pólya-problem.

Conjecture 8. The following functional computation problem can be solved in polynomial time: Given a square matrix A over $\{-1, 0, +1\}$, if A is not an SNS-matrix, then a matrix A' over $\mathbb{Q}$ with the same sign pattern as A' can be computed such that A' is singular.

3.2 Complement-Invariant Clause-Sets

By $\overline{\boldsymbol{F}} := \{\overline{C} : C \in F\}$ we denote the clause-wise complement of a clause-set F, where $\overline{\boldsymbol{C}} := \{\overline{x} : x \in C\}$). The following lemma shows that (un)satisfiability, minimal unsatisfiability and leanness of $F \cup \overline{F}$ is equivalent to the respective property for F where the full autarky system is replaced with the autarky system of balanced autarkies (other properties behave in a more complicated way; we consider only barely leanness here).

Lemma 9. *For a clause-set F we have:*

1. *For a partial assignment φ the following conditions are equivalent:*
   (a) *φ is an autarky for $F \cup \overline{F}$.*
   (b) *φ is a balanced autarky for $F \cup \overline{F}$.*
   (c) *φ is a balanced autarky for F.*
2. *F is balanced satisfiable resp. balanced unsatisfiable iff $F \cup \overline{F}$ is satisfiable resp. unsatisfiable.*
3. *F is balanced lean iff $F \cup \overline{F}$ is lean.*
4. *F is minimally balanced unsatisfiable iff $F \cup \overline{F}$ is minimally unsatisfiable.*
5. (a) *If F is barely balanced lean then $F \cup \overline{F}$ is barely lean.*
   (b) *Conversely, assume that $F \cup \overline{F}$ is barely lean.*
      i. *If $c(F) = 1$ (here the clause of F must be a unit clause), then F is not barely balanced lean.*
      ii. *If F is a generalised sum of some $F_1, \{U\}$, where U is a unit clause, then F is not barely balanced lean (since F_1 is balanced lean).*
      iii. *Otherwise F is barely balanced lean.*

Proof. Part 1 follows by definition, and Parts 2, 3 are direct consequences of this basic fact. For Part 4 it is left to show that if F is minimally balanced unsatisfiable then $F \cup \overline{F}$ is minimally unsatisfiable; consider a clause $C \in F \cup \overline{F}$, and we have to show that $(F \cup \overline{F}) \setminus \{C\}$ is satisfiable. There is a balanced satisfying assignment φ for $(F \cup \overline{F}) \setminus \{C, \overline{C}\}$. Since $F \cup \overline{F}$ is lean, φ touches C, and then φ or $\overline{\varphi}$ is a satisfying assignment for $(F \cup \overline{F}) \setminus \{C\}$. For Part 5 first assume that F is barely balanced lean, and we have to show that $F \cup \overline{F}$ is barely lean. Consider $C \in F \cup \overline{F}$. Now $(F \cup \overline{F}) \setminus \{C, \overline{C}\}$ has a non-trivial balanced autarky φ. Since $F \cup \overline{F}$ is lean, φ touches C, and then φ or $\overline{\varphi}$ (again) is a non-trivial autarky for $(F \cup \overline{F}) \setminus \{C\}$.

Conversely assume that $F \cup \overline{F}$ is barely lean, $c(F) \geq 2$, and consider $C \in F$. We know that F is balanced lean, and we need to consider whether it is barely so, that is, whether there is a non-trivial balanced autarky for $F \setminus \{C\}$. There is a non-trivial autarky φ for $(F \cup \overline{F}) \setminus \{C\}$. If φ touches $F \setminus \{C\}$, then φ is a non-trivial balanced autarky for $F \setminus \{C\}$. So assume $\mathrm{var}(\varphi) \cap \mathrm{var}(F \setminus \{C\}) = \emptyset$.

So there is $x \in C$ with $\mathrm{var}(x) \notin \mathrm{var}(F \setminus \{C\})$ (while $\mathrm{var}(C \setminus \{x\}) \subseteq \mathrm{var}(F \setminus \{C\})$, since F is lean), and thus F is a generalised sum of $F_1 := F \setminus \{C\}$ and $\{x\}$ (note that in case of $\mathrm{var}(F \setminus \{C\}) = \emptyset$ we would have $F \setminus \{C\} = \{\bot\}$). Now F_1 is balanced lean, since a non-trivial balanced autarky for $F \setminus \{C\}$ either does not touch C, or otherwise a non-balancedness can be repaired using x, and so in both cases we contradict that F is balanced lean. It follows be definition, (as stated in case 5(b)ii), that here F is not barely balanced lean. □

By Lemma 9, Part 4 together with Lemma 6 we get:

Corollary 10. *A clause-set $F \cup \overline{F}$, where F is a clause-set with $c(F) \geq 2$, is minimally unsatisfiable if and only if F is barely balanced lean and balanced autarky-indecomposable.*

Definition 11. *A clause-set F is called **complement-invariant** if $F = \overline{F}$, which is equivalent to the existence of a clause-set F_0 with $F = F_0 \cup \overline{F_0}$; such an F_0 is called a **core half** of F if $2 \cdot c(F_0) = c(F)$.*

So only complement-invariant clause-sets F with $\bot \notin F$ have a core half (which is unique only up to complementation of the clauses), but this little inconvenience seems not to justify the use of multi-clause-sets instead in this article (the problem is that $\overline{\bot} = \bot$, which causes contraction for clause-*sets*). If F is complement-invariant and φ is an autarky for F, then also $\varphi * F$ is complement-invariant (obviously this is not the case for arbitrary φ).

Definition 12. *For an arbitrary clause-set F we define the **reduced deficiency** $\boldsymbol{\delta_{\mathrm{r}}(F)} := \frac{1}{2}(\delta(F) - n(F)) \in \frac{1}{2}\mathbb{N}_0$.*

If F is complement-invariant with core half F_0, then we have $\delta_{\mathrm{r}}(F) = \delta(F_0)$, since

$$\delta_{\mathrm{r}}(F) = \frac{1}{2}(\delta(F) - n(F)) = \frac{1}{2}(c(F) - 2n(F)) = \frac{1}{2}(2c(F_0) - 2n(F)) = c(F_0) - n(F) = c(F_0) - n(F_0) = \delta(F_0).$$

If F is lean, then we have $\delta_{\mathrm{r}}(F) = \delta(F_0) \geq 0$ (see Lemma 19 for a more general statement).

3.3 Square Balanced Lean Clause-Sets

Theorem 13. *Consider a clause-set F with $\delta(F) = 0$.*

1. *It is decidable in poly-time whether F is balanced lean.*
2. *It is decidable in poly-time whether F is barely balanced lean.*
3. *Assume F is balanced lean. Then F is balanced autarky-decomposable if and only if F is a generalised sum of clause-sets F_1, F_2 with $\delta(F_1) = \delta(F_2) = 0$.*
4. *It is decidable in poly-time whether F is balanced autarky-indecomposable.*
5. *It is decidable in poly-time whether F is minimally balanced unsatisfiable.*

Proof. Part 1 follows by [13] as discussed in Subsection 3.1. For Part 2 first we test whether F is balanced lean (by Part 1); assume now that F is balanced lean, and consider $F \in C$. We have to test whether $F \setminus \{C\}$ is (not) balanced lean. If $\delta(F \setminus \{C\}) = 0$, then we can apply Part 1, and so assume $\delta(F \setminus \{C\}) \neq 0$. If C would contain two or more variables not occurring in $F \setminus \{C\}$ then F would not be balanced lean, and so all variables of C must occur in $F \setminus \{C\}$, and we have $\delta(F \setminus \{C\}) = -1$, in which case $F \setminus \{C\}$ is not balanced lean (as desired). The statement of Part 3 is essentially equivalent to Theorem 2.2.1 in [2], however for the sake of completeness, and also since our notions differ slightly from [2] in order to accommodate for the differences in handling matrices and clause-sets, we give a proof here. If F is balanced autarky-decomposable, then F is a generalised sum of balanced lean clause-sets F_1, F_2; we then have $\delta(F_1), \delta(F_2) \geq 0$, and due to $\delta(F) = \delta(F_1) + \delta(F_2)$ we get $\delta(F_1) = \delta(F_2) = 0$. For the opposite direction assume that F is a generalised sum of clause-sets F_1, F_2 with $\delta(F_1) = \delta(F_2) = 0$; we have that F_2 is balanced lean, and we show that F_1 is also balanced lean, which is equivalent to $M(F_1)$ being an SNS-matrix. We have the matrix decomposition

$$M(F) = \begin{pmatrix} M(F_1) & 0 \\ * & M(F_2) \end{pmatrix}.$$

Because of $\det(M(F)) = \det(M(F_1)) \cdot \det(M(F_2))$ we get $\det(M_1) \neq 0$, and moreover, if there would be two non-null terms in the determinant expansion of $M(F_1)$ then we would get two non-null terms in the determinant expansion of $M(F)$ contradicting that $M(F)$ is an SNS-matrix (see the characterisation of SNS-matrices in Subsection 3.1). For Part 4 w.l.o.g. we can assume that $\bot \notin F$; it suffices now to realise that F is a generalised sum of clause-sets F_1, F_2 with $\delta(F_1) = \delta(F_2) = 0$ iff $M(F)$ is partly decomposable as defined in Subsection 4.2 of [1] (there the square submatrices in the decomposition are allowed to be zero matrices, however this cannot happen in our case, since $M(F)$ has no zero column as a clause-variable matrix, while it has no zero row by assumption), where the property of being partly decomposable is decidable in polynomial time.[3] Finally Part 5 follows with Lemma 6 and Parts 2, 4 (and using that $\top$ is not minimally balanced unsatisfiable while $\{C\}$ for some unit clause C is minimally balanced unsatisfiable). □

Corollary 14. *Consider a complement-invariant clause-set F with $\delta_r(F) = 0$.*

1. *It is decidable in poly-time whether F is lean.*
2. *It is decidable in poly-time whether F is barely lean.*
3. *It is decidable in poly-time whether F is minimally unsatisfiable.*

[3] By Corollary 4.2.4 in [1] thus the algorithm for deciding whether F is balanced autarky-indecomposable works as follows: Let $A := |M(F)|$, and consider the bipartite graph G with reduced adjacency matrix A. If G has no perfect matching then F is balanced autarky-decomposable; otherwise obtain A' from A by permuting accordingly rows and columns of A in such a way that the main diagonal of A' has only entries equal to 1. Now F is balanced autarky-indecomposable iff the directed graph with adjacency matrix A' is strongly connected.

Proof. Part 1 follows with Theorem 13, Part 1 and Lemma 9, Part 3. Part 2 follows with Theorem 13, Part 2 and Lemma 9, Part 5 as follows: W.l.o.g. $\bot \notin F$. Let F_0 be a core half of F. If F_0 is not balanced lean, then F is not barely lean; assume now that F_0 is balanced lean. If F_0 is barely balanced lean, so is F; assume now that F_0 is not barely balanced lean. If $F_0 = \{C\}$, then F is barely lean iff C is a unit clause. If F_0 is not a generalised sum of $F'_0, \{C\}$ for some unit clause C, then F is not barely lean. The remaining case is that F_0 is a generalised sum of $F'_0, \{C\}$ for some unit clause C. Since F_0 is balanced lean, also F'_0 must be balanced lean here; now F is barely lean iff F'_0 is barely balanced lean. Finally Part 3 follows with Theorem 13, Part 5 and Lemma 9, Part 4. □

Five examples of minimally unsatisfiable complement-invariant clause-sets F with $\delta_r(F) = 0$ follow, where in each case only a core half is given, and variables are represented by positive integers (the first three examples yield PN-clause-sets, where every clause is either positive or negative):

1. $F_0^1 := \big\{ \{1,2\}, \{2,3\}, \{1,3\} \big\}$ (the 2-colourability of the triangle).
2. $F_0^2 := \big\{ \{2,3,6\}, \{1,3,5\}, \{1,2,4\}, \{3,4,7\}, \{2,5,7\}, \{1,6,7\}, \{4,5,6\} \big\}$ (the 2-colourability of the Fano plane (the projective plane of order 2)).
3. $F_0^3 := \big\{ \{1,2,3\}, \{2,3,4\}, \{1,4\}, \{2,5\}, \{3,5\} \big\}$ (an example with clauses with empty intersection).
4. $F_0^4 := \big\{ \{-1,2\}, \{-2,3,4\}, \{-1,-2,-3,4\}, \{-1,-4\} \big\}$ (an example where the whole clause-set cannot be renamed to a PN-clause-set (i.e., clashes within F_0 are necessary)).
5. $F_0^5 := \big\{ \{-1,2,-5\}, \{-1,-2,3\}, \{-2,-3,4\}, \{-3,-4,-5\}, \{1,-5\} \big\}$ (with clauses with empty variable-intersection and necessary clashes within F_0).

4 Linear Autarkies and Balanced Linear Autarkies

A *linear autarky* for a clause-set F is given by a non-trivial solution to the linear-programming problem $M(F)\cdot x \geq 0$, while a *balanced linear autarky* for F is given by a non-trivial solution to the linear algebra problem $M(F) \cdot x = 0$; see [5,7] for more information. Balanced linear autarkies are special balanced autarkies, and a partial assignment φ is a balanced linear autarky for F iff $\varphi, \overline{\varphi}$ are linear autarkies for F. The autarky-existence problem is solvable in polynomial time for linear autarkies as well as for balanced linear autarkies, and thus with Lemma 3 the basic decision problems for these two (normal) autarky systems are solvable in polynomial time (while it remains to determine the complexity of (balanced) linear-autarky-decomposability decision). In full analogy to Lemma 9 we have, replacing "lean" by "linearly lean" in the statements as well as in the proofs:

Lemma 15. *For a clause-set F we have:*

1. *For a partial assignment φ the following conditions are equivalent:*
   (a) *φ is a linear autarky for $F \cup \overline{F}$.*
   (b) *φ is a balanced linear autarky for $F \cup \overline{F}$.*
   (c) *φ is a balanced linear autarky for F.*

2. *F is balanced linearly satisfiable resp. balanced linearly unsatisfiable iff $F \cup \overline{F}$ is linearly satisfiable resp. linearly unsatisfiable.*
3. *F is minimally balanced linearly unsatisfiable iff $F \cup \overline{F}$ is minimally linearly unsatisfiable.*
4. *F is balanced linearly lean iff $F \cup \overline{F}$ is linearly lean.*
5. *(a) If F is barely balanced linearly lean then $F \cup \overline{F}$ is barely linearly lean.*
   (b) Conversely, assume that $F \cup \overline{F}$ is barely linearly lean.
      i. If $c(F) = 1$ (here the clause of F must be a unit clause), then F is not barely balanced linearly lean.
      ii. If F is a generalised sum of some $F_1, \{U\}$, where U is a unit clause, then F is not barely balanced linearly lean (since F_1 is balanced lean).
      iii. Otherwise F is barely balanced linearly lean.

By Lemma 6 we get (analogously to Corollary 10):

Corollary 16. *A clause-set $F \cup \overline{F}$ for $c(F) \geq 2$ is minimally linearly unsatisfiable if and only if F is barely balanced linearly lean and balanced linear-autarky-indecomposable.*

Analogously to Theorem 13, Parts 3, 4 we get (note that F with $\delta(F) = 0$ is balanced linearly lean iff $M(F)$ is non-singular):

Lemma 17. *Consider a clause-set F with $\delta(F) = 0$. If F is balanced linearly lean, then F is balanced linear-autarky-decomposable if and only if F is a generalised sum of clause-sets F_1, F_2 with $\delta(F_1) = \delta(F_2) = 0$. Thus it is decidable in poly-time whether F is balanced linear-autarky-indecomposable.*

The *maximal deficiency* (see [7]) is defined as $\delta^*(F) = \max_{F' \subseteq F} \delta(F) \geq 0$ (note that $\delta(\top) = 0$). Analogously we define:

Definition 18. *The **maximal reduced deficiency** is defined as* $\boldsymbol{\delta^*_{\mathrm{r}}(F)} := \max_{F' \subseteq F} \delta_{\mathrm{r}}(F') \geq 0$.

If F is complement-invariant and F_0 is a core half of F, then $\delta^*_{\mathrm{r}}(F) = \delta^*(F_0)$, and thus for complement-invariant clause-sets $\delta^*_{\mathrm{r}}(F)$ is computable in polynomial time. By Lemma 7.2 in [8] we have:

Lemma 19. *If a clause-set F_0 is balanced linearly lean, then $\delta^*(F_0) = \delta(F_0)$ (and thus $\delta(F_0) \geq 0$). It follows that for a linearly lean complement-invariant clause-set F we have $\delta^*_{\mathrm{r}}(F) = \delta_{\mathrm{r}}(F)$ (and thus $\delta_{\mathrm{r}}(F) \geq 0$).*

Theorem 20. *Under the condition that Conjecture 8 holds, for complement-invariant clause-sets F with $\delta^*_{\mathrm{r}}(F) = 0$ the lean kernel is computable in polynomial time (together with a maximal autarky realising the lean kernel). Thus the satisfiability problem is decidable in polynomial time (providing also a satisfying assignment in the satisfiable case), and furthermore if F is unsatisfiable, then a minimally unsatisfiable sub-clause-set can be computed in polynomial time.*

Proof. First the input F is reduced to the linearly lean kernel $F' \subseteq F$, where we have $\delta_{\mathrm{r}}(F') = 0$. If F' is lean, then F' is the lean kernel of F; otherwise by Conjecture 8 (and Lemma 9, Part 1) we can find a non-trivial autarky φ for F', and reduce F' to $F'' \subseteq F$. The whole cycle is repeated until we find the lean kernel of F. □

5 Hypergraph Colouring

Considering hypergraphs G (that is, pairs $G = (V(G), E(G))$, where $V(G)$ is a (finite) set of vertices, and $E(G)$ is the hyperedge set, a set of subsets of $V(G)$) as special (positive) clause-sets, we naturally transfer the notion of deficiency:

Definition 21. *For a hypergraph G let the* ***deficiency*** *be defined as* $\boldsymbol{\delta(G)} := |E(G)| - |V(G)|$*, while the* ***maximal deficiency*** *is* $\boldsymbol{\delta^*(G)} := \max_{G' \subseteq G} \delta(G)$*, where by "$G' \subseteq G$" we denote sub-hypergraphs of G, that is, hypergraphs G' with $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$.*

The maximal deficiency of a hypergraph is the maximum size of a matching in the bipartite hyperedge-vertex graph, and thus is computable in polynomial time. Core halves of complement-invariant PN-clause-sets F (where a PN-clause-set contains only positive and negative clauses, and no mixed cases) can be naturally considered as hypergraphs $G(F)$, which actually can be defined for arbitrary clause-sets as the **variable hypergraph**, that is $V(G(F)) = \mathrm{var}(F)$, while $E(G(F)) = \{\mathrm{var}(C) : C \in F\}$. It is easy to see that a complement-invariant PN-clause-set F is satisfiable resp. minimally unsatisfiable if and only if $G(F)$ is 2-colourable resp. minimally non-2-colourable (see Lemma 8.1 in [8] for the general statement, regarding k-colouring of hypergraphs), and thus from Theorem 20 we directly obtain:

Corollary 22. *Under the condition that Conjecture 8 holds, the 2-colouring problem for hypergraphs G with $\delta^*(G) = 0$ can be solved in polynomial time.*

A very informative classification of minimally non-2-colourable *intersecting* square hypergraphs has been given in [14], allowing to replace the complicated algorithm underlying Corollary 22 by some form of simple pattern matching (two central examples F_0^1, F_0^2 have been given after Corollary 14); see [8] for more details (and for the interpretation of these hypergraphs by minimally unsatisfiable complement-invariant clause-sets of reduced deficiency zero with the property, that every two different clauses have some variable in common).

6 Open Problems

The main open problem seems to me the following generalisation (the case $k = 0$ is Theorem 20, but relying on Conjecture 8):

Conjecture 23. For fixed $k \in \mathbb{N}_0$ the satisfiability problem for complement-invariant clause-sets F with $\delta_{\mathrm{r}}^*(F) \leq k$ is decidable in polynomial time (and in the satisfiable cases also a satisfying assignment can be computed).

One can further ask whether we have fixed-parameter tractability in k here. Conjecture 23 implies that for fixed k the hypergraph 2-colouring problem is decidable in polynomial time for hypergraphs G with $\delta^*(G) \leq k$; now what about m-colourability for arbitrary $m \geq 2$? What is the resolution complexity of minimally unsatisfiable complement-invariant clause-sets F with $\delta_{\mathrm{r}}(F) = 0$

($\delta_r(F) = k$) ? Finally, we express the hope, that the broader context of boolean as well as non-boolean satisfiability will help to gain a better understanding also for the original combinatorial problems (like finding a Pfaffian orientation of a (non-bipartite) graph; see [15]).

References

1. Richard A. Brualdi and Herbert J. Ryser. *Combinatorial Matrix Theory*, volume 39 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1991. ISBN 0-521-32265-0; QA188.B78 1991.
2. Richard A. Brualdi and Bryan L. Shader. *Matrices of sign-solvable linear systems*, volume 116 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 1995. ISBN 0-521-48296-8; QA188.B79.
3. Herbert Fleischner, Oliver Kullmann, and Stefan Szeider. Polynomial–time recognition of minimal unsatisfiable formulas with fixed clause–variable difference. *Theoretical Computer Science*, 289(1):503–516, November 2002.
4. Oliver Kullmann. An application of matroid theory to the SAT problem. In *Fifteenth Annual IEEE Conference on Computational Complexity (2000)*, pages 116–124.
5. Oliver Kullmann. Investigations on autark assignments. *Discrete Applied Mathematics*, 107:99–137, 2000.
6. Oliver Kullmann. On the use of autarkies for satisfiability decision. In Henry Kautz and Bart Selman, editors, *LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, volume 9 of *Electronic Notes in Discrete Mathematics (ENDM)*. Elsevier Science, June 2001.
7. Oliver Kullmann. Lean clause-sets: Generalizations of minimally unsatisfiable clause-sets. *Discrete Applied Mathematics*, 130:209–249, 2003.
8. Oliver Kullmann. Constraint satisfaction problems in clausal form: Autarkies, minimal unsatisfiability, and applications to hypergraph inequalities. In Nadia Creignou, Phokion Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints*, number 06401 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. `http://drops.dagstuhl.de/opus/volltexte/2006/803`.
9. Oliver Kullmann, Inês Lynce, and João Marques-Silva. Categorisation of clauses in conjunctive normal forms: Minimally unsatisfiable sub-clause-sets and the lean kernel. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, volume 4121 of *Lecture Notes in Computer Science*, pages 22–35. Springer, 2006. ISBN 3-540-37206-7.
10. Oliver Kullmann, Victor W. Marek, and Mirosław Truszczyński. Computing autarkies and properties of the autarky monoid. In preparation, January 2007.
11. William McCuaig. Pólya's permanent problem. *The Electronic Journal of Combinatorics*, 11, 2004. #R79, 83 pages.
12. Christos H. Papadimitriou and David Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37:2–13, 1988.
13. Neil Robertson, Paul D. Seymour, and Robin Thomas. Permanents, Pfaffian orientations, and even directed circuits. *Annals of Mathematics*, 150:929–975, 1999.
14. Paul D. Seymour. On the two-colouring of hypergraphs. *The Quarterly Journal of Mathematics (Oxford University Press)*, 25:303–312, 1974.
15. Robin Thomas. A survey of Pfaffian orientations of graphs. In *Proceedings of the International Congress of Mathematicians, Madrid, Spain*, 2006.

Verifying Propositional Unsatisfiability: Pitfalls to Avoid

Allen Van Gelder

University of California, Santa Cruz CA 95060, USA
http://www.cse.ucsc.edu/~avg

Abstract. The importance of producing a certificate of unsatisfiability is increasingly recognized for high performance propositional satisfiability solvers. The leading solvers develop a conflict graph as the basis for deriving (or "learning") new clauses. Extracting a resolution derivation from the conflict graph is theoretically straightforward, but it turns out to have some surprising practical pitfalls (as well as the unsurprising problem that resolution proofs can be extremely long). These pitfalls are exposed, solutions are presented, and analyzed for worse cases. Dramatic improvements on industrial benchmarks are demonstrated.

1 Introduction

With the explosive growth of Sat Modulo Theories (SMT) in the last few years, the focus in propositional SAT solvers is shifting to unsatisfiable formulas, because these are the negated theorems to be proved in many applications. Producing proofs and independently checking them has received limited attention. Two ground-breaking efforts are Goldberg and Novikov [4], who built on BerkMin [3], and Zhang and Malik [9,10], who built on Chaff [6]. It is important to get our propositional house in order to provide an adequate foundation for the more sophisticated challenge of producing independently checkable proofs for SMT.

The author has argued elsewhere [7] that solvers should be able to produce *easily verifiable* certificates to support claims of unsatisfiability. The gold standard proposed is that the language of certificates should be recognizable in *deterministic log space*, a very low complexity class. Intuitively, an algorithm to recognize a log space language may re-read the input as often as desired, but can only write into working storage consisting of a fixed number of registers, each able to store $O(\log L)$ bits, for inputs of length L.

The rationale for such a stringent requirement is that the buck has to stop somewhere. How are we to trust a "verifier" that is far too complex to be subjected to an automated verification system? And how are we to trust that automated verification system? Eventually, there has to be a verifier that is so elementary that we are satisfied with human inspection.

An explicit resolution proof is one in which each derived clause is stated explicitly, along with the two earlier clauses that were resolved to get the current clause. It is not hard to see that an explicit resolution proof can be recognized in deterministic log space.

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 328–333, 2007.

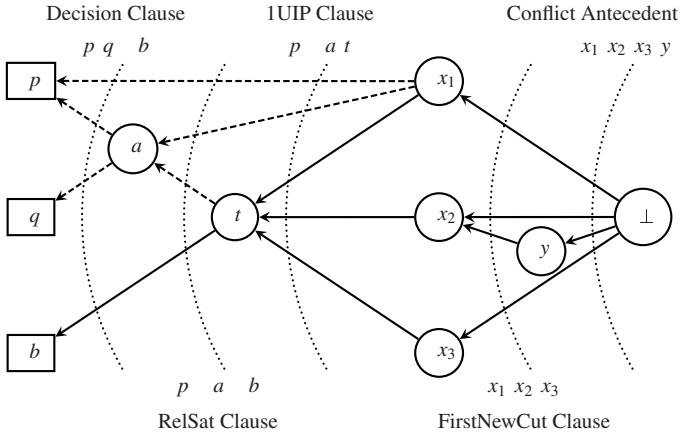


Fig. 1. Conflict graph with several cuts shown

A detailed specification for an explicit resolution derivation (`%RES`) was used for the "certified unsat" track of the SAT 2005 Competition. Although the results of that track were disappointing, this paper shows how fixing a performance problem in one solver produced orders-of-magnitude improvement, suggesting that the methodology is feasible, after all. Details on current capabilities are available at: `http://www.cse.ucsc.edu/~avg/ProofChecker/`.

2 A Pitfall in Resolution Extraction

Most, if not all, leading SAT solvers use a *conflict graph* data structure to infer *conflict clauses*. Figure 1 illustrates a conflict graph. The notation varies from other papers [11,2] to better reflect the actual data structures used by the programs. Each graph vertex is associated with a different literal, no complementary literals appear, and the conflict vertex is associated with the constant *false*, denoted by "⊥." The vertex for each implied literal, including *false*, is labeled with an "input" clause, called the *antecedent clause*. This notation agrees more closely with the original presentation . Assumed (guessed) literals, commonly called "decision literals," do not have an antecedent clause.

Recent papers have observed the connection between conflict graphs and resolution [4,9,2,8]. Of course, given a cut, the antecedent clauses on the conflict side logically imply the conflict clause, which consists of the negations of the reason-side literals adjacent to some vertex on the conflict side (i.e., one or more edges cross the cut to such literals).

The question is how to exploit the structure of the conflict graph to obtain a resolution derivation of the conflict clause. This question is not as simple as it might appear, in view of the fact that the algorithm published by Zhang and Malik, and actually implemented in `zverify_df` (in the `zchaff` distributions), has an exponential worst case. Their algorithm is based on Figure 3 *of their*

```
1.    cl = final_conflicting_clause;
2.    while (!is_empty_clause(cl)) {
3.        lit = choose_literal(cl);
4.        var = variable_of_literal(lit);
5.        ante_cl = antecedent(var);
6.        cl = resolve(cl, ante_cl); }
```

Fig. 2. Pseudocode to generate resolution proof from conflict graph

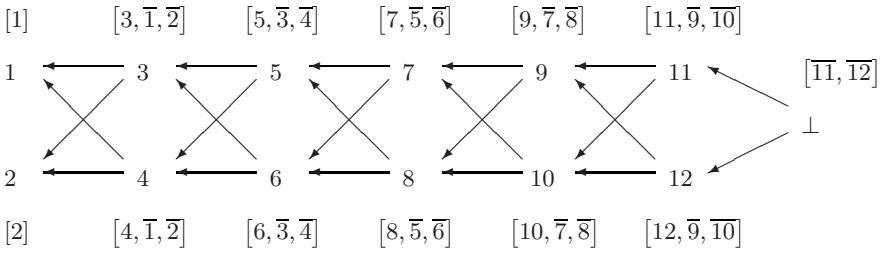


Fig. 3. DAG family ($h = 6$ instance) with exponential worst case for `zverify_df` as published and distributed (through 2006). Antecedent clauses are shown in brackets.

paper [9], the crucial part of which appears in *our* Figure 2. It is important to note on line 1 that `final_conflicting_clause` is *not* a conflict clause. Rather, it denotes the antecedent of $\bot$, the "input" clause that became empty during unit propagation (see "Conflict Antecedent" in Figure 1). The *conflict clause* is the final value of `cl`, and is empty if the solver was correct. An invariant is that `cl` contains the *negations of* some literals in the conflict graph.

Line 3 is implemented two ways in different versions. In one version, the literal chosen is one with minimum *DAG height*, which is the maximum path length to a vertex whose antecedent is a unit clause. In another version, the literal chosen is simply the one with the lowest variable number, essentially an arbitrary choice. The conflict graph family that generates exponential behavior in the size of the conflict graph is the same for both versions. The parameter of this family is h, the DAG height of the *false* vertex. A member with $h = 6$ is shown in Figure 3. The resolution developed by Figure 2 begins as follows, where "(11)" denotes resolution with clashing literal 11:

$$[\overline{11}, \overline{12}]\ (11)\ [11, \overline{9}, \overline{10}]\ (9)\ [9, \overline{7}, \overline{8}]\ (7)\ [7, \overline{5}, \overline{6}]\ (5)\ [5, \overline{3}, \overline{4}]\ (3)\ [3, \overline{1}, \overline{2}]$$
$$(1)\ [1]\ (2)\ [2]\ (4)\ [4, \overline{1}, \overline{2}]\ (1)\ [1]\ (2)\ [2]\ (6)\ [6, \overline{3}, \overline{4}]\ (3)\ [3, \overline{1}, \overline{2}]\ \ldots$$

Literals 1 and 2 will be resolved upon 2^{h-1} times each; literals 3 and 4 will be resolved upon 2^{h-2} times each, etc.; the total number of resolutions is $2\,(2^h - 1)$.

This is a case where theory translates into practice, at least in the case of the arbitrary choice of literal in the version dated 2004.11.15. Table 1 shows the sizes of resolution proofs on some smaller industrial benchmarks, for the

Table 1. Resolution proof length inefficiencies. Sizes are mega-literals. "+?" indicates job was killed when size exceeded stated number.

Small GN03		
benchmark	2004.11.15	with fix
5pipe	153	15
5pipe_1_ooo	986	92
5pipe_5_ooo	2320	94
6pipe	169	97
6pipe_6_ooo	3072+?	486
7pipe	358	319
9vliw_bp_mc	308	58
barrel7	2754	5
barrel8	3072+?	73
barrel9	3072+?	80
c3540	393	78
c5315	162	9
c7552	2650	22

IBM_FV SAT 2005		
benchmark	2004.11.15	with fix
01_SAT_dat.k10	3	0.134
07_SAT_dat.k30	3	2.582
07_SAT_dat.k35	3	2.859
18_SAT_dat.k10	174	0.511
18_SAT_dat.k15	102520+?	14.410
1_11_SAT_dat.k10	13	0.338
1_11_SAT_dat.k15	128	2.077
20_SAT_dat.k10	13	0.208
23_SAT_dat.k10	3	0.062
23_SAT_dat.k15	15259+?	3.737
26_SAT_dat.k10	0.036	0.036
2_14_SAT_dat.k10	70	0.857
2_14_SAT_dat.k15	3220	7.188

original program and a fixed version that uses a better order, which will be described shortly. (Although `zverify_df` does not output the resolution proof, it materializes all the clauses.) The GN03 benchmarks are the smaller ones reported previously [4,9]. Those labeled "`IBM_FV`" are from the "industrial" category of the SAT 2005 Competition (see `http://www.satcompetition.org/2005` and the URL given in the introduction).

The purpose of showing this data is to show that extracting a resolution proof from the conflict graph has the potential for very bad performance, but it is not intended to criticize the Princeton `zchaff` team in any way, who have been very cooperative. Their paper *did* go through peer review without the problem being noticed.

3 Avoiding the Pitfall Via "Trivial Resolution" (TVR)

We now introduce some terminology and notation to study efficient methods of extraction, with worst-case guarantees. We shall use the notation that n is the number of vertices in the conflict graph, m is the number of edges, d is the maximum out-degree of any vertex in the conflict graph, and w is the number of literals in the conflict clause to be derived. Note that the sum of the lengths of the antecedent clauses is m. Thus $(m + w)$ is the combined length of the input and output.

A *linear resolution derivation* is one in which the first clause is an "input" clause, called the *top clause*, and each resolution operation has the previous clause in the derivation as one operand; the second operand may be an "input" clause or an earlier-derived clause of the linear derivation. (We shorten "resolution derivation" to "derivation" when there is no ambiguity.) An *input derivation* is a linear derivation in which the second operand must be an "input" clause. For our purposes, an

"input" clause is one that existed while the conflict graph was being constructed; this includes antecedent clauses of the conflict graph. Note that Figure 2 provides a framework for input derivations using antecedent clauses.

Beame *et al.* [2] define a *trivial resolution derivation* (**TVR**) to be an input derivation with the further restriction that no clashing variable occurs in more than one resolution operation. They show (their Proposition 4) that the conflict clause can be derived by a TVR using the antecedent clauses of the conflict-side vertices of the conflict graph, and using the antecedent of the *false* vertex as the top clause. The successful TVR has exactly n resolution operations, but using a correct order is crucial.

To get a worst-case bound on total derivation size of TVR, measured in number of literals, we note that it is possible for the current resolvent to grow by up to $(d-1)$ literals per resolution for the first n/d steps to a size of $(w+1+n\,(d-1)/d)$, even if the final conflict clause is fairly narrow. Thereafter, it can shrink only one literal per resolution, so the sum of the sizes of all resolvents in the derivation is bounded by $(w + n\,(d-1)/2\,d)\,n$, which can be quadratic in the size of the conflict graph.

The trouble with TVR scheme is that the correct order is not readily accessible from the data structure of the conflict graph. To obtain a valid TVR order, the rule for `choose_literal` on line 3 of Figure 2 should be:

Cut-Crossing Rule: Choose a literal all of whose incoming edges originate from a vertex whose literal has already been resolved upon, or from the *false* vertex (reworded from Beame *et al., op cit.*).

Some other data structure is needed to provide or compute an appropriate order.

The saving grace is that the input for `zverify_df` is set up by the companion solver, such as `zchaff`, which already *has* such a data structure. The solver creates a sequence of "implied" literals in the chronological order in which they entered the conflict graph. When this sequence is available, `zchaff` uses the *reverse* of this order to generate the trace of an input derivation for `zverify_df` to verify [9]. It is not difficult to see that reverse chronological order satisfies the cut-crossing rule. (A theoretical nitpick is that the sequence might include numerous implied literals that are not in the conflict graph, but have to be looked at anyway, so the *time* is not strictly bounded by the size parameters of the conflict graph.)

As it happens, `zchaff` communicates the successful order to `zverify_df` in its encoding of the "resolve-trace" for all conflict clauses with a *positive* "decision level." Unfortunately, `zchaff` (through 2006) treats decision level zero differently, does not actually create an empty conflict clause, and so the crucial order for decision level zero is *not* in the encoding of the resolve-trace. Instead the final conflict graph itself is encoded in the output. Thus `zverify_df` had an opportunity to go astray.

The "fix" applied by this author to deliver the rightmost column of Table 1 (files available from the author; meanwhile, the "diffs" are available at the URL mentioned in the introduction) involved modifying `zchaff` to follow through at decision level zero, mainly using the procedures and data structures already in

the program. The fixed program creates an empty conflict clause and encodes its resolution order in the resolve-trace, using the same protocol as for the nonempty conflict clauses at positive decision levels. Then `zverify_df` was modified slightly to expect this additional line.

4 Conclusion

A longer paper at the URL in the introduction has additional details, experimental results, and discusses another extraction strategy named Pseudo-Unit Propagation (PUP). Both the TVR and PUP methods guarantee that total derivation size polynomial in the size of the conflict graph, but both have nonlinear worst cases, which are incomparable. Experimental data on industrial benchmarks (not presented in detail) shows that PUP derivations are 60% longer than TVR on average and are longer on about 74% of the benchmarks tested.

The significance of this data and take-home message is: The PUP strategy is a second, milder, pitfall to avoid. A program that generates resolutions "on-line" during unit clause propagation will do essentially the same resolutions as a PUP after-the-fact system. Both theoretical and empirical analyses suggest that this produces more verbose resolution derivations (aside from the on-line resolutions that turn out to be unneeded), compared to the after-the-fact TVR method.

References

1. Baase, S., Van Gelder, A.: Computer Algorithms: Introduction to Design and Analysis. 3rd edn. Addison-Wesley (2000)
2. Beame, P., Kautz, H., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. J. Artificial Intelligence Research **22** (2004) 319–351
3. Goldberg, E., Novikov, Y.: Berkmin: a fast and robust sat-solver. In: Proc. Design, Automation and Test in Europe. (2002) 142–149
4. Goldberg, E., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: Proc. Design, Automation and Test in Europe. (2003) 886–891
5. Marques-Silva, J.P., Sakallah, K.A.: GRASP–a search algorithm for propositional satisfiability. IEEE Transactions on Computers **48** (1999) 506–521
6. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: 39th Design Automation Conference. (2001)
7. Van Gelder, A.: Decision procedures should be able to produce (easily) checkable proofs. In: CP02 Workshop on Constraints in Formal Verification, Ithaca. (2002)
8. Van Gelder, A.: Pool resolution and its relation to regular resolution and DPLL with clause learning. In: Proc. LPAR, LNAI 3835, Montego Bay. (2005) 580–594
9. Zhang, L., Malik, S.: Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In: Proc. Design, Automation and Test in Europe. (2003)
10. Zhang, L., Malik, S.: Extracting small unsatisfiable cores from unsatisfiable boolean formula. In: Proc. Theory and Applications of Satisfiability Testing. (2003)
11. Zhang, L., Madigan, C., Moskewicz, M., Malik, S.: Efficient conflict driven learning in a boolean satisfiability solver. In: ICCAD. (2001)

A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories

Alessandro Cimatti[1], Alberto Griggio[2], and Roberto Sebastiani[2]

[1] ITC-IRST, Povo, Trento, Italy
cimatti@itc.it
[2] DIT, Università di Trento, Italy
{griggio,rseba}@dit.unitn.it

Abstract. Finding small unsatisfiable cores for SAT problems has recently received a lot of interest, mostly for its applications in formal verification. Surprisingly, the same problem in the context of SAT Modulo Theories (SMT) has instead received very little attention in the literature; in particular, we are not aware of any work aiming at producing small unsatisfiable cores in SMT.

The purpose of this paper is to start filling the gap in this area, by proposing a novel approach for computing small unsat cores in SMT. The main idea is to combine an SMT solver with an external propositional core extractor: the SMT solver produces the theory lemmas found during the search; the core extractor is then called on the boolean abstraction of the original SMT problem and of the theory lemmas. This results in an unsatisfiable core for the original SMT problem, once the remaining theory lemmas have been removed.

The approach has several advantages: it is extremely simple to implement and to update, and it can be interfaced with every propositional core extractor in a plug-and-play way, so that to benefit for free of all unsat-core reduction techniques which have been or will be made available.

1 Motivations and Goals

In the last decade we have witnessed an impressive advance in the efficiency of SAT techniques, which has brought large and previously intractable problems at the reach of state-of-the-art SAT solvers. In particular, and due to its importance in formal verification, the problem of finding small *unsatisfiable cores* in SAT — i.e., unsatisfiable subsets of unsatisfiable sets of clauses— has been addressed by many authors in the recent years [8,9,11,4,7,10].

The formalism of plain propositional logic, however, is often not suitable or expressive enough for representing many interesting real-world problems, which are more naturally expressible as satisfiability problems in decidable first-order theories —Satisfiability Modulo Theories, SMT. Efficient SMT solvers have been developed in the last five years, called *lazy* SMT solvers, which combine DPLL with ad-hoc decision procedures for many theories of interest (e.g., [6,1,2,5]).

Surprisingly, the problem of finding unsatisfiable cores in SMT has received virtually no attention in the literature. Although some SMT tools do compute

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 334–339, 2007.

unsat cores, this is done either as a byproduct of the more general task of producing proofs, or by modifying the embedded DPLL solver so that to apply basic propositional techniques to produce an unsat core. In particular, we are not aware of any work aiming at producing *small* unsatisfiable cores in SMT.

In this paper we present a novel approach addressing this problem. The main idea is to combine an SMT solver with an external propositional core extractor. The SMT solver stores and returns the theory lemmas it had to prove in order to refute the input formula; the external core extractor is then called on the boolean abstraction of the original SMT problem and of the theory lemmas. The resulting boolean unsatisfiable core is cleaned from (the boolean abstraction of) all theory lemmas, and it is refined back into a subset of the original clauses. The result is an unsatisfiable core of the original SMT problem.

Although simple in principle, the approach is conceptually interesting: basically, the SMT solver is used to dynamically lift the suitable amount of theory information to the boolean level. Furthermore, the approach has several advantages in practice: first, it is extremely simple to implement and to update; second, it is effective in finding small cores; third, the core extraction is not prone to complex SMT reasoning; finally, it can be interfaced with every propositional core extractor in a plug-and-play manner, so that to benefit for free of all unsat-core reduction techniques which have been or will be made available.

For lack of space, in this short version of the paper we omit many details, any related work and the description and the results of our extensive experimental evaluation of the approach. They can be found in the extended version of the paper [3].

2 Background

Given a decidable first-order theory $\mathcal{T}$, we call a *theory solver for* $\mathcal{T}$, *$\mathcal{T}$-solver*, any tool able to decide the satisfiability in $\mathcal{T}$ of sets/conjunctions of ground atomic formulas and their negations (*theory literals* or *$\mathcal{T}$-literals*) in the language of $\mathcal{T}$. If the input set of $\mathcal{T}$-literals μ is $\mathcal{T}$-unsatisfiable, then a typical *$\mathcal{T}$-solver* not only returns unsat, but it also returns the subset η of $\mathcal{T}$-literals in μ which was found $\mathcal{T}$-unsatisfiable. (η is hereafter called a *theory conflict set*, and $\neg\eta$ a *theory conflict clause*.) If μ is $\mathcal{T}$-satisfiable, then *$\mathcal{T}$-solver* not only returns sat, but it may also be able to discover one (or more) deductions in the form $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$, s.t. $\{l_1, ..., l_n\} \subseteq \mu$ and l is an unassigned $\mathcal{T}$-literal. If so, we call $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ a *theory-deduction clause*. Importantly, notice that both theory-conflict clauses and theory-deduction clauses are valid in $\mathcal{T}$. We call them *theory lemmas* or *$\mathcal{T}$-lemmas*.

Satisfiability Modulo (the) Theory $\mathcal{T}$ ($SMT(\mathcal{T})$) is the problem of deciding the satisfiability of *boolean combinations* of propositional atoms and theory atoms. We call an $SMT(\mathcal{T})$ *tool* any tool able to decide $SMT(\mathcal{T})$. Notice that, unlike a *$\mathcal{T}$-solver*, an $SMT(\mathcal{T})$ tool must handle also boolean connectives.

Hereafter we adopt the following terminology and notation. The bijective function $\mathcal{T}2\mathcal{P}$ ("theory-to-propositional"), called *boolean abstraction*, maps propositional variables into themselves, ground $\mathcal{T}$-atoms into fresh propositional

variables, and is homomorphic w.r.t. boolean operators and set inclusion. The function $\mathcal{P}2\mathcal{T}$ ("propositional-to-theory"), called *refinement*, is the inverse of $\mathcal{T}2\mathcal{P}$. The symbols φ, ψ denote $\mathcal{T}$-formulas, and μ, η denote sets of $\mathcal{T}$-literals; φ^p, ψ^p denote propositional formulas, μ^p, η^p denote sets of propositional literals (i.e., truth assignments) and we often use them as synonyms for the boolean abstraction of φ, ψ, μ, and η respectively, and vice versa (e.g., φ^p denotes $\mathcal{T}2\mathcal{P}(\varphi)$, μ denotes $\mathcal{P}2\mathcal{T}(\mu^p)$). If $\mathcal{T}2\mathcal{P}(\varphi) \models \bot$, then we say that φ is *propositionally unsatisfiable*.

2.1 Lazy Techniques for SMT

The idea underlying every lazy $SMT(\mathcal{T})$ procedure is that (a complete set of) the truth assignments for the propositional abstraction of φ are enumerated and checked for satisfiability in $\mathcal{T}$; the procedure either returns sat if one $\mathcal{T}$-satisfiable truth assignment is found, or returns unsat otherwise.

A simplified schema of a lazy $SMT(\mathcal{T})$ procedure is as follows. The propositional abstraction φ^p of the input formula φ is given as input to a modified version of a DPLL solver, and when a satisfying assignment μ^p is found, the refinement μ of μ^p is fed to the *$\mathcal{T}$-solver*; if μ is found $\mathcal{T}$-consistent, then φ is $\mathcal{T}$-consistent; otherwise, *$\mathcal{T}$-solver* returns the conflict set η which caused the $\mathcal{T}$-inconsistency of $\mathcal{P}2\mathcal{T}(\mu^p)$. Then the clause $\neg\eta^p$ is added in conjunction to φ^p, either temporarily or permanently (*$\mathcal{T}$-learning*), and the algorithm backtracks up to the highest point in the search where a literal can be unit-propagated on $\neg\eta^p$ (*$\mathcal{T}$-backjumping*).

Two important optimizations are *early pruning* and *theory propagation*: the $\mathcal{T}$-solver is invoked also on (the refinement of) an intermediate assignment μ: if it is found $\mathcal{T}$-unsatisfiable, then the procedure can backtrack, since no extension of μ can be $\mathcal{T}$-satisfiable; if not, and if the *$\mathcal{T}$-solver* performs a deduction $\{l_1, ..., l_n\} \models_{\mathcal{T}} l$ s.t. $\{l_1, ..., l_n\} \subseteq \mu$, then $\mathcal{T}2\mathcal{P}(l)$ can be unit-propagated, and the boolean abstraction of the $\mathcal{T}$-lemma $(\bigvee_{i=1}^{n} \neg l_i \vee l)$ can be learned.

The above schema is a coarse abstraction of the procedures underlying all the state-of-the-art lazy $SMT(\mathcal{T})$ tools like, e.g., BarceLogic, CVCLite, MathSAT, Yices. The interested reader is pointed to, e.g., [6,1,2,5], for details and further references.

2.2 Techniques for Unsatisfiable-Core Extraction in SAT

Given an unsatisfiable (propositional) CNF formula φ, we say that an unsatisfiable CNF formula ψ is an *unsatisfiable core* of φ iff $\varphi = \psi \wedge \psi'$ for some (possibly empty) CNF formula ψ'. Intuitively, ψ is a subset of the clauses in φ causing the unsatisfiability of φ. An unsatisfiable core ψ is *minimal* iff the formula obtained by removing any of the clauses of ψ is satisfiable. A *minimum* unsat core is a minimal unsat core with the smallest possible cardinality.

In the last few years, several algorithms for computing small [11], minimal [7,4] or minimum [8,9,10] unsatisfiable cores of propositional formulas have been proposed. For lack of space, we can not provide details here, and we refer to the extended version [3] of the paper for a detailed description.

```
⟨SatValue,Clause_set⟩ T-Unsat_Core(Clause_set φ) { // φ is {C_1, ..., C_n}
  if (T-DPLL(φ) == sat) then return ⟨sat,∅⟩;
  // D_1, ..., D_k are the T-lemmas stored by T-DPLL
  ψ^p=Boolean_Unsat_Core(T2P({C_1, ..., C_n, D_1, ..., D_k}));
  // ψ^p is T2P({C'_1, ..., C'_m, D'_1, ..., D'_j}));
  return ⟨unsat,{C'_1, ..., C'_m}⟩; }
```

Fig. 1. Schema of the $\mathcal{T}$-Unsat_Core algorithm

2.3 Techniques for Unsatisfiable-Core Extraction in SMT

To the best of our knowledge, there is no published work in the literature devoted to the computation of unsatisfiable cores in SMT. However, at least three SMT solvers support unsat core generation with techniques adapted from SAT. CVCLite [1] and a recent extension of MathSAT [2] can compute unsatisfiable cores as a byproduct of the generation of proofs, in a way similar to that in [11]. Yices [5] instead uses the following technique: a selector variable is introduced for each original clause, which is forced to false before starting the search. In this way, when a conflict at decision level zero is found, the conflict clause contains only selector variables, and the unsat core returned is the union of the clauses whose selectors appear in such conflict clause.

We remark the fact that none of these solvers aims at producing minimal or minimum unsat cores, nor does anything to reduce their size.

3 A Novel Approach to Building Unsat Cores in SMT

We present a novel approach in which the unsatisfiable core is computed *a posteriori* w.r.t. the execution of the SMT solver, and only if the formula has been found $\mathcal{T}$-unsatisfiable, by means of an external (and possibly optimized) propositional unsat-core extractor.

In the following we assume that a lazy $SMT(\mathcal{T})$ procedure has been run over a $\mathcal{T}$-unsatisfiable $SMT(\mathcal{T})$ CNF formula $\varphi =_{def} \{C_1, ..., C_n\}$, and that $D_1, ..., D_k$ denote all the $\mathcal{T}$-lemmas, both theory-conflict and theory-deduction clauses, which have been returned by the $\mathcal{T}$*-solver* during the run.

Our novel approach is based on two simple facts.

(i) Under the assumptions above, the conjunction of φ with all the $\mathcal{T}$-lemmas $D_1, ..., D_k$ is propositionally unsatisfiable: $\mathcal{T}2\mathcal{P}(\varphi \wedge \bigwedge_{i=1}^{n} D_i) \models \bot$.

(ii) As $\mathcal{T}$-lemmas D_i are valid in $\mathcal{T}$, they do not affect the $\mathcal{T}$-satisfiability of a formula: $(\psi \wedge D_i) \models_{\mathcal{T}} \bot \iff \psi \models_{\mathcal{T}} \bot$.

These facts suggest the novel algorithm represented in Figure 1. The procedure $\mathcal{T}$`-Unsat_Core` receives as input a set of clauses $\varphi =_{def} C_1, ..., C_n$ and it invokes on it a lazy $SMT(\mathcal{T})$ tool $\mathcal{T}$`-DPLL`, which is instructed to *store* somewhere the $\mathcal{T}$-lemmas returned by $\mathcal{T}$*-solver*, namely $D_1, ..., D_k$. If $\mathcal{T}$`-DPLL` returns `sat`, then the whole procedure returns `sat`. Otherwise, the boolean abstraction

of $\{C_1, ..., C_n, D_1, ..., D_k\}$, which is inconsistent because of (i), is passed to an external tool `Boolean_Unsat_Core`, which is able to return the boolean unsat core ψ^p of the input. By construction, ψ^p is the boolean abstraction of a clause set $\{C'_1, ..., C'_m, D'_1, ..., D'_j\}$ s.t. $\{C'_1, ..., C'_m\} \subseteq \{C_1, ..., C_n\}$ and $\{D'_1, ..., D'_j\} \subseteq \{D_1, ..., D_k\}$. As ψ^p is unsatisfiable, then $\{C'_1, ..., C'_m, D'_1, ..., D'_j\}$ is $\mathcal{T}$-unsat. By (ii), the $\mathcal{T}$-valid clauses $D'_1, ..., D'_j$ have no role in the $\mathcal{T}$-unsatisfiability of $\{C'_1, ..., C'_m, D'_1, ..., D'_j\}$, so that the procedure returns `unsat` and the $\mathcal{T}$-unsat core $\{C'_1, ..., C'_m\}$.

The procedure can be implemented very simply by modifying the SMT solver so that to store the $\mathcal{T}$-lemmas[1] —if it doesn't already— and by interfacing it with some state-of-the-art boolean unsat-core extractor used as an external black-box device (e.g., by a simple exchange of files in DIMACS format). Moreover, if the SMT solver can provide the set of all $\mathcal{T}$-lemmas as output, then the whole procedure may reduce to a control device interfacing with both the SMT solver and the boolean core extractor as black-box external devices.

3.1 Discussion

Though based on an extremely simple concept, the newly-proposed approach is appealing for several reasons.

First, it is extremely simple to implement and update. The building of unsat cores is demanded to an external device, which is fully decoupled from the internal DPLL-based enumerator. Therefore, there is no need to implement any internal unsat-core constructor nor to modify the embedded boolean device. Every possible external device can be interfaced in a plug-and-play manner by simply exchanging a couple of DIMACS files.

Second, from the perspective of effectiveness in reducing the size of unsat cores, every original clause which the boolean unsat-core device is able to drop is dropped also in the final formula. Therefore, this technique exploits for free all unsat-core reduction techniques which have been and will be conceived in the SAT community.

One potential drawback of this approach is the fact that a $SMT(\mathcal{T})$ solver is required to store all the $\mathcal{T}$-lemmas returned by the $\mathcal{T}$-*solver*. However, this is not a real problem. In fact, unlike with plain SAT, in lazy SMT the computational effort is typically dominated by the search in the theory $\mathcal{T}$, so that the number of clauses that can be stored with a reasonable amount of memory is typically much bigger than the number of calls to the $\mathcal{T}$-*solver* which can overall be accomplished within a reasonable amount of time. In our experience, even the hardest SMT formulas at the reach of current lazy SMT solvers rarely need generating more than 10^5 $\mathcal{T}$-lemmas, which have very reasonable memory requirements to store. (E.g., notice that the default choice in MathSAT is to learn all $\mathcal{T}$-lemmas permanently anyway, and we have never encountered memory overload problems due to this fact.)

[1] Notice that here "storing" does not mean "learning": the SMT solver is not required to add the $\mathcal{T}$-lemmas to the formula during the search. This imposes no constraint on the lazy strategy adopted.

Finally, one limitation of this approach is that the resulting $\mathcal{T}$-unsatisfiable core is not guaranteed to be minimal, even if `Boolean_Unsat_Core` returns minimal boolean unsatisfiable cores. However, to the best of our knowledge, not only the issue of the *minimality* of unsat cores in SMT has never been addressed or even discussed before, but also this is the first time that the problem of the *size* of unsat cores in SMT is addressed.

4 Conclusions

We have presented a novel approach to generating small unsatisfiable cores in SMT, that computes them a posteriori, relying on an external propositional unsat core extractor. The technique is very simple in concept, and straightforward to implement and update. Moreover, it benefits for free of all the advancements in propositional unsat core computation. Our experimental results, available in the extended version [3], have shown that, by using different core extractors, it is possible to reduce significantly the size of cores and to trade core quality for speed of execution (and vice versa), with no implementation effort.

References

1. C. Barrett and S. Berezin. CVC Lite: A New Implementation of the Cooperating Validity Checker. In *Proc. CAV'04*, 2004.
2. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P.van Rossum, S. Schulz, and R. Sebastiani. An incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In *Proc. TACAS'05*, 2005.
3. A. Cimatti, A. Griggio, and R. Sebastiani. A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories. Technical Report DIT-07-006, DIT, Univ. of Trento, 2007. Extended version. Available at `http://dit.unitn.it/~griggio/papers/sat07_extended.pdf`.
4. N. Dershowitz, Z. Hanna, and A. Nadel. A Scalable Algorithm for Minimal Unsatisfiable Core Extraction. In *Proc. SAT'06*, 2006.
5. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc. CAV'06*, 2006.
6. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *Proc. CAV'04*, 2004.
7. R. Gershman, M. Koifman, and O. Strichman. Deriving Small Unsatisfiable Cores with Dominators. In *Proc. CAV'06*, 2006.
8. I. Lynce and J. P. Marques Silva. On computing minimum unsatisfiable cores. In *Proc. SAT'04*, 2004.
9. M. N. Mneimneh, I. Lynce, Z. S. Andraus, J. P. Marques Silva, and K. A. Sakallah. A Branch-and-Bound Algorithm for Extracting Smallest Minimal Unsatisfiable Formulas. In *Proc. SAT'05*, 2005.
10. J. Zhang, S. Li, and S. Shen. Extracting Minimum Unsatisfiable Cores with a Greedy Genetic Algorithm. In *Proc. ACAI'06*, 2006.
11. L. Zhang and S. Malik. Extracting small unsatisfiable cores from unsatisfiable boolean formula. In *Proc. SAT'03*, 2003.

SAT Solving for Termination Analysis with Polynomial Interpretations*

Carsten Fuhs[1], Jürgen Giesl[1], Aart Middeldorp[2], Peter Schneider-Kamp[1], René Thiemann[1], and Harald Zankl[2]

[1] LuFG Informatik 2, RWTH Aachen, Germany
{fuhs,giesl,psk,thiemann}@informatik.rwth-aachen.de
[2] Institute of Computer Science, University of Innsbruck, Austria
{aart.middeldorp,harald.zankl}@uibk.ac.at

Abstract. Polynomial interpretations are one of the most popular techniques for automated termination analysis and the search for such interpretations is a main bottleneck in most termination provers. We show that one can obtain speedups in orders of magnitude by encoding this task as a SAT problem and by applying modern SAT solvers.

1 Introduction

Termination is one of the most important properties of programs and therefore, there is a need for techniques and tools that analyze the termination behavior of programs automatically. In particular, there has been intensive research on methods for termination analysis of *term rewrite systems* (TRSs) [4]. Instead of developing several separate termination techniques for different programming languages, a promising approach is to transform programs from different languages into TRSs instead. Then termination tools for TRSs can be used for termination analysis of many different programming languages, cf. e.g. [13,22].

The increasing interest in termination analysis for TRSs is also shown by the annual *International Competition of Termination Tools*.[1] In 2006, for the first time some tools used SAT solvers to automate certain termination techniques, cf. [1,5,6,11,18,25,26]. But although *polynomial interpretations* [20] are one of the most popular techniques in these tools, up to now there has not been any paper on using SAT solvers for finding polynomial interpretations automatically.

In this paper, we show that SAT solving is extremely useful for this task. We recapitulate TRSs in Sect. 2. Sect. 3 shows how to encode the search for polynomial interpretations as a SAT problem. Sect. 4 extends our approach to *negative* polynomial interpretations [17]. Sect. 5 presents our implementation in the tool AProVE [14], which was the most powerful termination prover for TRSs in all the competitions 2004 - 2006. Our experiments show that our approach improves dramatically over previous methods for generating polynomial interpretations.

* Supported by the DFG (Deutsche Forschungsgemeinschaft) grant GI 274/5-1 and the FWF (Austrian Science Fund) project P18763.

[1] See http://www.lri.fr/~marche/termination-competition/

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 340–354, 2007.

2 Termination of TRSs and Polynomial Interpretations

A TRS $\mathcal{R}$ is a set of rules $\ell \to r$ where ℓ and r are terms. A rule $\ell \to r$ applies to a term t if ℓ matches a subterm u of t with some substitution σ (namely, $u = \sigma(\ell)$). The rule is applied by replacing the subterm u by $\sigma(r)$, resulting in a new term v (a so-called *rewrite step*, denoted "$t \to_{\mathcal{R}} v$"). A *reduction* is a sequence of rewrite steps. A TRS is *terminating* if all its reductions are finite. For example, consider the following TRS where s represents the successor function, $\mathsf{half}(x)$ computes $\lfloor \frac{x}{2} \rfloor$, and $\mathsf{bits}(x)$ is the number of bits needed to represent all numbers up to x.

$$\begin{array}{rl@{\qquad}rl} \mathsf{half}(0) \to 0 & \text{(i)} & \mathsf{bits}(0) \to 0 & \text{(iv)} \\ \mathsf{half}(\mathsf{s}(0)) \to 0 & \text{(ii)} & \mathsf{bits}(\mathsf{s}(0)) \to \mathsf{s}(0) & \text{(v)} \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{half}(x)) & \text{(iii)} & \mathsf{bits}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{bits}(\mathsf{s}(\mathsf{half}(x)))) & \text{(vi)} \end{array}$$

So we have $\mathsf{half}(\mathsf{s}(\mathsf{s}(0))) \to_{\mathcal{R}} \mathsf{s}(\mathsf{half}(0)) \to_{\mathcal{R}} \mathsf{s}(0)$, i.e., $\mathsf{half}(\mathsf{s}(\mathsf{s}(0))) \to^*_{\mathcal{R}} \mathsf{s}(0)$.

One of the most powerful termination methods is the *dependency pair* (DP) technique [2], implemented in virtually all current termination tools for TRSs.

Definition 1 (Dependency Pairs [2]). *For a TRS $\mathcal{R}$, the* defined *symbols are the root symbols of the left-hand sides of rules. For every defined symbol f, we extend the signature by a fresh* tuple symbol *$f^\sharp$ with the same arity as f. If $t = f(t_1, \ldots, t_n)$ and f is a defined symbol, we write $t^\sharp$ for $f^\sharp(t_1, \ldots, t_n)$. If $\ell \to r \in \mathcal{R}$ and t is a subterm of r with defined root symbol, then the rule $\ell^\sharp \to t^\sharp$ is a* dependency pair *of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted $DP(\mathcal{R})$.*

In our example, half and bits are defined symbols and $DP(\mathcal{R}) = \{\text{(vii)}, \text{(viii)}, \text{(ix)}\}$:

$$\begin{array}{rl@{\qquad}rl} \mathsf{half}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}^\sharp(x) & \text{(vii)} & & \\ \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}^\sharp(x) & \text{(viii)} & \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{bits}^\sharp(\mathsf{s}(\mathsf{half}(x))) & \text{(ix)} \end{array}$$

Intuitively, a DP corresponds to a (possibly recursive) function call. To prove termination, we have to show that there cannot be infinitely many function calls in any reduction. More precisely, one has to prove that there is no infinite chain

$$\sigma_1(u_1) \to_{DP(\mathcal{R})} \sigma_1(v_1) \;\to^*_{\mathcal{R}}\; \sigma_2(u_2) \to_{DP(\mathcal{R})} \sigma_2(v_2) \;\to^*_{\mathcal{R}}\; \sigma_3(u_3) \to_{DP(\mathcal{R})} \sigma_3(v_3) \;\ldots$$

where $u_i \to v_i \in DP(\mathcal{R})$ and σ_i are substitutions. To this end, the DP method[2] requires $u \succ v$ for all $u \to v \in DP(\mathcal{R})$ and $\ell \succsim r$ for all rules $\ell \to r \in \mathcal{R}$:

$$\bigwedge\nolimits_{u \to v \,\in\, DP(\mathcal{R})} \quad u \succ v \qquad \wedge \qquad \bigwedge\nolimits_{\ell \to r \,\in\, \mathcal{R}} \quad \ell \succsim r \tag{1}$$

A popular method to search for relations $\succ$ and $\succsim$ automatically are *polynomial interpretations* [20]. A polynomial interpretation $\mathcal{P}ol$ maps each n-ary function symbol f to a polynomial $f_{\mathcal{P}ol}$ over n variables $x_1, \ldots, x_n$ with coefficients from $\mathbb{N} = \{0, 1, 2, \ldots\}$. This mapping is extended to terms by defining $[x]_{\mathcal{P}ol} = x$ for all variables x and $[f(t_1, \ldots, t_n)]_{\mathcal{P}ol} = f_{\mathcal{P}ol}([t_1]_{\mathcal{P}ol}, \ldots, [t_n]_{\mathcal{P}ol})$. If the interpretation $\mathcal{P}ol$ is clear from the context, we also write $[t]$ instead of $[t]_{\mathcal{P}ol}$.

[2] For further refinements of the DP method we refer to [2,12,15,16,17], for example.

For example, consider $\mathcal{P}ol_1$ with $\mathsf{half}_{\mathcal{P}ol_1} = \mathsf{half}^\sharp_{\mathcal{P}ol_1} = x_1$, $\mathsf{bits}_{\mathcal{P}ol_1} = \mathsf{bits}^\sharp_{\mathcal{P}ol_1} = \mathsf{s}_{\mathcal{P}ol_1} = x_1 + 1$, $\mathsf{0}_{\mathcal{P}ol_1} = 0$. Then $[\mathsf{half}(\mathsf{s}(\mathsf{s}(x)))] = x + 2$ and $[\mathsf{s}(\mathsf{half}(x))] = x + 1$. Now a term u is considered to be greater (resp. greater-equal) than v iff $[u] > [v]$ (resp. $[u] \geq [v]$) holds for all instantiations of the variables with natural numbers. So with $\mathcal{P}ol_1$ we obtain $\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{s}(\mathsf{half}(x))$. In fact, all DPs (vii) - (ix) are strictly decreasing and the rules (i) - (vi) are at least weakly decreasing, i.e., the requirement (1) holds. Thus, termination of the TRS (i) - (vi) is proved.

To find such interpretations automatically, one starts with an *abstract* polynomial interpretation. It maps each n-ary symbol f to a polynomial of the form

$$a_0 + a_1\, x_1^{e_{11}} \dots x_n^{e_{n1}} \;+\; \dots \;+\; a_m\, x_1^{e_{1m}} \dots x_n^{e_{nm}} \tag{2}$$

Here, the e_{ij} are actual numbers (i.e., one has to determine the degree and the shape of the polynomials), but the coefficients a_i are left open (i.e., they are *variable* or *abstract* coefficients). For example, we could use the abstract polynomial interpretation $\mathcal{P}ol_2$ with $\mathsf{half}_{\mathcal{P}ol_2} = a\,x_1 + b$, $\mathsf{s}_{\mathcal{P}ol_2} = c\,x_1 + d$, etc.

Every inequality $u \succ v$ (resp. $u \succsim v$) can be transformed into the constraint $[u] - [v] > 0$ (resp. $[u] - [v] \geq 0$). Here, $[u] - [v]$ is a polynomial of the form

$$p_0 + p_1\, x_1^{e_{11}} \dots x_n^{e_{n1}} + \dots + p_k\, x_1^{e_{1k}} \dots x_n^{e_{nk}} \tag{3}$$

where p_i are polynomials over abstract coefficients. So with $\mathcal{P}ol_2$, $\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{s}(\mathsf{half}(x))$ is transformed to $a\,c^2\,x + a\,c\,d \;+ a\,d + b - c\,a\,x - c\,b - d \;>\; 0$, i.e. to

$$p_0 + p_1\,x \;>\; 0 \quad \text{where} \quad p_0 = a\,c\,d \;+ a\,d + b - c\,b - d \quad \text{and} \quad p_1 = a\,c^2 - c\,a \tag{x}$$

If p is a polynomial like (3), then instead of inequalities or equalities of the form $p > 0$, $p \geq 0$, $p = 0$, it suffices[3] to require the following constraints [19]:

$$\alpha_{p>0} \;=\; (p_0 > 0 \wedge p_1 \geq 0 \wedge \dots \wedge p_k \geq 0) \tag{4}$$
$$\alpha_{p\geq 0} \;=\; (p_0 \geq 0 \wedge p_1 \geq 0 \wedge \dots \wedge p_k \geq 0) \tag{5}$$
$$\alpha_{p=0} \;=\; (p_0 = 0 \wedge p_1 = 0 \wedge \dots \wedge p_k = 0) \tag{6}$$

So instead of (x), it is sufficient to demand $p_0 > 0$ and $p_1 \geq 0$:

$$a\,c\,d \;+ a\,d + b - c\,b - d > 0 \qquad \wedge \qquad a\,c^2 - c\,a \;\geq\; 0 \tag{xi}$$

Such constraints can be transformed further such that they do not contain subtractions and "$\geq$" anymore. For example, (xi) can be transformed into

$$a\,c\,d \;+ a\,d + b > c\,b + d \;\;\wedge\;\; (a\,c^2 > c\,a \;\;\vee\;\; a\,c^2 = c\,a) \tag{xii}$$

Now to prove termination one has to show the *satisfiability* of such *Diophantine constraints* over the naturals. Def. 2 introduces their syntax and semantics.

Definition 2 (Diophantine Constraints). *Let $\mathcal{A}$ be a set of Diophantine variables. The set of* polynomials *$\mathcal{P}$ is the smallest set with*

- $\mathcal{A} \subseteq \mathcal{P}$ *and* $\mathbb{N} \subseteq \mathcal{P}$
- *If* $\{p,\, q\} \subseteq \mathcal{P}$ *then* $\{p + q,\; p * q\} \subseteq \mathcal{P}$

[3] Of course, $\alpha_{p>0}$ and $\alpha_{p\geq 0}$ are sufficient, but not necessary for $p > 0$ and $p \geq 0$.

The set of Diophantine constraints $\mathcal{C}$ *is the smallest set with*

- $\{true, false\} \subseteq \mathcal{C}$
- *If* $\{p,\ q\} \subseteq \mathcal{P}$ *then* $\{p > q,\ p = q\} \subseteq \mathcal{C}$
- *If* $\{\alpha,\ \beta\} \subseteq \mathcal{C}$ *then* $\{\neg\alpha,\ \alpha \wedge \beta,\ \alpha \vee \beta,\ \alpha \to \beta,\ \alpha \leftrightarrow \beta,\ \alpha \oplus \beta\} \subseteq \mathcal{C}$

A Diophantine interpretation $\mathcal{D}$ *is a mapping* $\mathcal{D} : \mathcal{A} \to \mathbb{N}$. *It can be extended to polynomials by defining* $\mathcal{D}(n) = n$ *for all* $n \in \mathbb{N}$, $\mathcal{D}(p + q) = \mathcal{D}(p) + \mathcal{D}(q)$, *and* $\mathcal{D}(p * q) = \mathcal{D}(p) * \mathcal{D}(q)$. *It can also be extended to Diophantine constraints as follows (i.e., we then have* $\mathcal{D} : \mathcal{C} \to \{0, 1\}$, *where* 0 *stands for "false" and* 1 *stands for "true"). As usual,* $\mathcal{D}$ *is called a* model *of a constraint* α *iff* $\mathcal{D}(\alpha) = 1$.

- $\mathcal{D}(true) = 1$, $\mathcal{D}(false) = 0$
- $\mathcal{D}(p > q) = 1$ *if* $\mathcal{D}(p) > \mathcal{D}(q)$ *and* $\mathcal{D}(p > q) = 0$, *otherwise*
- $\mathcal{D}(p = q) = 1$ *if* $\mathcal{D}(p) = \mathcal{D}(q)$ *and* $\mathcal{D}(p = q) = 0$, *otherwise*
- $\mathcal{D}(\neg\alpha) = 1$ *if* $\mathcal{D}(\alpha) = 0$ *and* $\mathcal{D}(\neg\alpha) = 0$, *otherwise,*
  and similarly for the other Boolean connectives, where $\oplus$ *is exclusive-or*

For example, let $a \in \mathcal{A}$ and let $\mathcal{D}$ with $\mathcal{D}(a) = 2$. Then $\mathcal{D}(2 * a) = \mathcal{D}(2) * \mathcal{D}(a) = 2 * 2 = 4$ and $\mathcal{D}(1 + a) = 3$. Thus, $\mathcal{D}(2 * a > 1 + a) = 1$, since $4 > 3$.

Similarly, the constraint (xii) is satisfied by the interpretation $\mathcal{D}(a) = 1$, $\mathcal{D}(b) = 0$, $\mathcal{D}(c) = 1$, and $\mathcal{D}(d) = 1$. This Diophantine interpretation instantiates the abstract polynomial interpretation $\mathcal{P}ol_2$ with $\mathsf{half}_{\mathcal{P}ol_2} = a\,x_1 + b$ and $\mathsf{s}_{\mathcal{P}ol_2} = c\,x_1 + d$ to the concrete polynomial interpretation $\mathcal{P}ol_1$ with $\mathsf{half}_{\mathcal{P}ol_1} = x_1$ and $\mathsf{s}_{\mathcal{P}ol_1} = x_1 + 1$ (i.e., we also write[4] $\mathcal{D}(\mathcal{P}ol_2) = \mathcal{P}ol_1$).

To summarize, to prove termination we proceed as follows:

1. Transform the termination problem into inequalities $u \succ v$ or $u \succsim v$ between terms. If one uses the DP method, then one obtains a requirement like (1).
2. Fix an abstract polynomial interpretation and transform the inequalities into $[u] - [v] > 0$ or $[u] - [v] \geq 0$, respectively.
3. Replace $[u] - [v] > 0$ and $[u] - [v] \geq 0$ by $\alpha_{[u]-[v]>0}$ and $\alpha_{[u]-[v]\geq 0}$, cf. (4), (5).
4. Transform the obtained constraint into a Diophantine constraint containing only $>$ and $=$ and no subtractions.
5. Check the satisfiability of the resulting Diophantine constraint. In the next section, we will show how to perform this check using SAT solvers.

3 Encoding Diophantine Constraints to SAT

We have shown that to prove termination, it suffices to prove the satisfiability of a Diophantine constraint. Now we reduce this problem to a SAT problem. We first give the syntax and semantics of propositional logic. Here, we also regard *tuples* of formulas which are interpreted as binary representations of numbers.

Definition 3 (Propositional Logic). *Let* $\mathcal{V}$ *be a set of propositional variables. Then the set of propositional* formulas $\mathcal{F}$ *is the smallest set with*

[4] $\mathcal{D}$ only instantiates abstract coefficients like a, b, c, d. For variables x_i we define $\mathcal{D}(x_i) = x_i$. Thus $\mathcal{D}(a\,x_1 + b) = 1 * x_1 + 0 = x_1$.

- $\mathcal{V} \subseteq \mathcal{F}$ *and* $\{0,\ 1\} \subseteq \mathcal{F}$
- *If* $\{\varphi,\ \psi\} \subseteq \mathcal{F}$ *then* $\{\neg\varphi,\ \varphi \wedge \psi,\ \varphi \vee \psi,\ \varphi \to \psi,\ \varphi \leftrightarrow \psi,\ \varphi \oplus \psi\} \subseteq \mathcal{F}$

A propositional interpretation $\mathfrak{I} : \mathcal{V} \to \{0,1\}$ *can be extended to formulas as follows (i.e., we then have* $\mathfrak{I} : \mathcal{F} \to \{0,1\}$*).* $\mathfrak{I}$ *is called a* model *of* φ *iff* $\mathfrak{I}(\varphi) = 1$*.*

- $\mathfrak{I}(0) = 0$*,* $\mathfrak{I}(1) = 1$
- $\mathfrak{I}(\neg\varphi) = 1$ *if* $\mathfrak{I}(\varphi) = 0$ *and* $\mathfrak{I}(\neg\varphi) = 0$*, otherwise (similarly for* $\wedge, \vee, \to, \leftrightarrow, \oplus$*)*

Finally, a propositional interpretation can also be extended to tuples *of n propositional formulas (with* $n \geq 1$*) by defining* $\mathfrak{I} : \mathcal{F}^n \to \mathbb{N}$ *where*

$$\mathfrak{I}(\langle\varphi_1, \ldots, \varphi_n\rangle) = 2^{n-1} * \mathfrak{I}(\varphi_1) + 2^{n-2} * \mathfrak{I}(\varphi_2) + \ldots + 2 * \mathfrak{I}(\varphi_{n-1}) + \mathfrak{I}(\varphi_n)$$

As an example, let $a_1, a_2 \in \mathcal{V}$ with $\mathfrak{I}(a_1) = 1$ and $\mathfrak{I}(a_2) = 0$. Then we have $\mathfrak{I}(\langle a_1, \neg a_2 \wedge 1, a_2\rangle) = 4 * \mathfrak{I}(a_1) + 2 * \mathfrak{I}(\neg a_2 \wedge 1) + \mathfrak{I}(a_2) = 4 * 1 + 2 * 1 + 0 = 6$.

Note that one can always delete zeros at the beginning of a tuple since $\mathfrak{I}(\langle 0, \ldots, 0, \varphi_1, \ldots, \varphi_n\rangle) = \mathfrak{I}(\langle\varphi_1, \ldots, \varphi_n\rangle)$ for any interpretation $\mathfrak{I}$. Moreover, we identify one-element-tuples with the element itself since $\mathfrak{I}(\langle\varphi\rangle) = \mathfrak{I}(\varphi)$.

Satisfiability of Diophantine constraints is undecidable (it corresponds to Hilbert's 10th problem). Therefore, we restrict the search to Diophantine interpretations of the form $\mathcal{D} : \mathcal{A} \to \{0, \ldots, 2^k - 1\}$ for a fixed $k \geq 1$. Then variables are only instantiated by numbers that can be represented by k bits. Satisfiability of Diophantine constraints by such restricted interpretations is NP-complete.

We now introduce a mapping $||.|| : \mathcal{C} \to \mathcal{F}$ from Diophantine constraints to propositional formulas such that a constraint α is satisfiable by an interpretation $\mathcal{D} : \mathcal{A} \to \{0, \ldots, 2^k - 1\}$ iff the propositional formula $||\alpha||$ is satisfiable.

We first define $||.||$ on Diophantine variables. Every Diophantine variable is mapped to a tuple of k propositional variables, i.e., we have $||.|| : \mathcal{A} \to \mathcal{V}^k$:

$$||a|| = \langle a_1, \ldots, a_k\rangle \text{ for every Diophantine variable } a \in \mathcal{A} \tag{7}$$

The idea is that $\langle a_1, \ldots, a_k\rangle$ should be the binary representation of a. For any propositional interpretation $\mathfrak{I}$ we define the *corresponding* interpretation $\mathcal{D}_\mathfrak{I}$.

Definition 4 (Corresponding Interpretations). *Let* $\mathcal{V}$ *contain* $a_1, \ldots, a_k$ *for any Diophantine variable* $a \in \mathcal{A}$*. For any propositional interpretation* $\mathfrak{I}$*, we define the* corresponding Diophantine interpretation *as* $\mathcal{D}_\mathfrak{I}(a) = \mathfrak{I}(\langle a_1, \ldots, a_k\rangle)$*.*

So if $k = 2$, then $||a|| = \langle a_1, a_2\rangle$. The propositional interpretation $\mathfrak{I}(a_1) = 1$ and $\mathfrak{I}(a_2) = 0$ corresponds to the interpretation with $\mathcal{D}_\mathfrak{I}(a) = \mathfrak{I}(\langle a_1, a_2\rangle) = 2$.

Now we define $||.||$ for natural numbers. Again, $||.||$ maps numbers to their binary representation, i.e., we have $||.|| : \mathbb{N} \to \{0,1\}^+$:

$$||n|| = \langle b^1, \ldots, b^\ell\rangle \text{ for every } n \in \mathbb{N} \tag{8}$$

where all $b^i \in \{0,1\}$ and $n = 2^{\ell-1} * b^1 + 2^{\ell-2} * b^2 + \ldots + 2 * b^{\ell-1} + b^\ell$. To avoid unnecessary long encodings with zeros at the beginning, we require $b^1 = 1$ for all $n > 0$ (i.e., we require that as few bits as possible are used for representing $n > 0$). So for example, we have $||2|| = \langle 1, 0\rangle$. For the representation of the number 0 we define $||0|| = \langle 0\rangle$. Note that $\mathcal{D}_\mathfrak{I}(n) = n = \mathfrak{I}(||n||)$ for all $n \in \mathbb{N}$.

Next we define $||.||$ for polynomials. As before, every polynomial is mapped to a tuple of propositional formulas, i.e., $||.|| : \mathcal{P} \to \mathcal{F}^+$. The goal is to obtain the following correspondence for all polynomials p and all interpretations $\mathfrak{I}$:

$$\mathcal{D}_{\mathfrak{I}}(p) = \mathfrak{I}(||p||) \tag{9}$$

To handle addition and multiplication, we introduce operations $B^+ : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}^+$ and $B^* : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}^+$ on tuples of propositional formulas. We then define

$$||p + q|| = B^+(||p||, ||q||) \quad \text{and} \quad ||p * q|| = B^*(||p||, ||q||) \tag{10}$$

for all polynomials p and q. We first give the definition of B^+.

- $B^+(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle) = B^+(\langle\varphi_1, ..., \varphi_n\rangle, \langle \underbrace{0, ..., 0}_{n-m \text{ times}}, \psi_1, ..., \psi_m\rangle)$ if $n > m$
- $B^+(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle) = B^+(\langle \underbrace{0, ..., 0}_{m-n \text{ times}}, \varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle)$ if $n < m$
- $B^+(\langle\varphi\rangle, \langle\psi\rangle) = \langle\varphi \wedge \psi, \varphi \oplus \psi\rangle$
- $B^+(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_n\rangle) = \langle B^{2or3}(\varphi_1, \psi_1, \xi_1), B^{1or3}(\varphi_1, \psi_1, \xi_1), \xi_2, ..., \xi_n\rangle$ if $B^+(\langle\varphi_2, ..., \varphi_n\rangle, \langle\psi_2, ..., \psi_n\rangle) = \langle\xi_1, ..., \xi_n\rangle$

Thus, ξ_1 is the carry resulting from adding $\langle\varphi_2, ..., \varphi_n\rangle$ and $\langle\psi_2, ..., \psi_n\rangle$. Here "$B^{1or3}(\varphi_1, \psi_1, \xi_1)$" abbreviates $\varphi_1 \oplus \psi_1 \oplus \xi_1$ (i.e., either one or all three of the formulas φ_1, ψ_1, and ξ_1 must be true). Similarly, "$B^{2or3}(\varphi_1, \psi_1, \xi_1)$" abbreviates $(\varphi_1 \wedge \psi_1) \vee (\varphi_1 \wedge \xi_1) \vee (\psi_1 \wedge \xi_1)$. For example, we have[5]

$B^+(\langle 1\rangle, \langle a_2\rangle) = \langle 1 \wedge a_2, 1 \oplus a_2\rangle = \langle a_2, \neg a_2\rangle$
$B^+(\langle 0, 1\rangle, \langle a_1, a_2\rangle) = \langle B^{2or3}(0, a_1, a_2), B^{1or3}(0, a_1, a_2), \neg a_2\rangle = \langle a_1 \wedge a_2,\ a_1 \oplus a_2,\ \neg a_2\rangle$

Therefore, we obtain $||1 + a|| = B^+(||1||, ||a||) = B^+(\langle 1\rangle, \langle a_1, a_2\rangle) = \langle a_1 \wedge a_2,\ a_1 \oplus a_2,\ \neg a_2\rangle$. Indeed, if $\mathfrak{I}(a_1) = 1$ and $\mathfrak{I}(a_2) = 0$ (i.e., $\mathcal{D}_{\mathfrak{I}}(a) = 2$), then $\mathcal{D}_{\mathfrak{I}}(1 + a) = 3$ and $\mathfrak{I}(||1 + a||) = \mathfrak{I}(\langle a_1 \wedge a_2, a_1 \oplus a_2, \neg a_2\rangle) = 3$. Hence, $\mathcal{D}_{\mathfrak{I}}(1 + a) = \mathfrak{I}(||1 + a||)$, as desired in (9). Next we give the definition of $B^* : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}^+$.

- $B^*(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi\rangle) = \langle\varphi_1 \wedge \psi, ..., \varphi_n \wedge \psi\rangle$
- $B^*(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle) = B^+(\langle\varphi_1 \wedge \psi_1, ..., \varphi_n \wedge \psi_1, \underbrace{0, ..., 0}_{m-1 \text{ times}}\rangle,$ $B^*(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_2, ..., \psi_m\rangle)\ \ ),$ if $m \geq 2$.

E.g., $||2 * a|| = B^*(||2||, ||a||) = B^*(\langle 1, 0\rangle, \langle a_1, a_2\rangle)$
$= B^+(\langle 1 \wedge a_1, 0 \wedge a_1, 0\rangle, B^*(\langle 1, 0\rangle, \langle a_2\rangle)) = B^+(\langle a_1, 0, 0\rangle, \langle a_2, 0\rangle)$
$= B^+(\langle a_1, 0, 0\rangle, \langle 0, a_2, 0\rangle) = \langle 0, a_1, a_2, 0\rangle = \langle a_1, a_2, 0\rangle.$

Indeed, if $\mathfrak{I}(a_1) = 1$ and $\mathfrak{I}(a_2) = 0$ (i.e., $\mathcal{D}_{\mathfrak{I}}(a) = 2$), then $\mathcal{D}_{\mathfrak{I}}(2 * a) = 4 = \mathfrak{I}(\langle a_1, a_2, 0\rangle) = \mathfrak{I}(||2 * a||)$, as desired in (9). We state (9) as a general lemma.

Lemma 5 (Correctness of Encoding Polynomials). *For every polynomial $p \in \mathcal{P}$ and every propositional interpretation $\mathfrak{I}$, we have $\mathcal{D}_{\mathfrak{I}}(p) = \mathfrak{I}(||p||)$.*[6]

[5] For readability, we perform Boolean simplifications like replacing $1 \wedge a_2$ by a_2, etc.
[6] All proofs can be found in [10].

Now we extend the mapping $||.||$ to $||.|| : \mathcal{C} \to \mathcal{F}$. Thus, every Diophantine constraint is mapped to a formula (not to a tuple). Obviously, we define

$$||\mathit{true}|| = 1 \quad \text{and} \quad ||\mathit{false}|| = 0 \tag{11}$$

For Diophantine constraints that are polynomial inequalities or equalities, we introduce operations $B^{>} : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}$ and $B^{=} : \mathcal{F}^+ \times \mathcal{F}^+ \to \mathcal{F}$ and define

$$||p > q|| = B^{>}(||p||, ||q||) \quad \text{and} \quad ||p = q|| = B^{=}(||p||, ||q||) \tag{12}$$

for all polynomials p and q. To define $B^{>}$ and $B^{=}$, we first handle the case where the argument tuples have different lengths. For $\circ \in \{=, >\}$ we define

- $B^{\circ}(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle) = B^{\circ}(\langle\varphi_1, ..., \varphi_n\rangle, \langle \underbrace{0, ..., 0}_{n-m \text{ times}}, \psi_1, ..., \psi_m\rangle)$ if $n > m$
- $B^{\circ}(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle) = B^{\circ}(\langle \underbrace{0, ..., 0}_{m-n \text{ times}}, \varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle)$ if $n < m$

Now we define $B^{>}$ and $B^{=}$ for tuples of equal length.

- $B^{=}(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_n\rangle) = (\varphi_1 \leftrightarrow \psi_1) \wedge \ldots \wedge (\varphi_n \leftrightarrow \psi_n)$
- $B^{>}(\langle\varphi\rangle, \langle\psi\rangle) = \varphi \wedge \neg\psi$
- $B^{>}(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_n\rangle) = (\varphi_1 \wedge \neg\psi_1) \vee ((\varphi_1 \leftrightarrow \psi_1) \wedge B^{>}(\langle\varphi_2, ..., \varphi_n\rangle, \langle\psi_2, ..., \psi_n\rangle))$, if $n \geq 2$

For example,
$$\begin{aligned} ||2 * a > 1 + a|| &= B^{>}(||2 * a||, ||1 + a||) \\ &= B^{>}(\langle a_1, a_2, 0\rangle, \langle a_1 \wedge a_2, a_1 \oplus a_2, \neg a_2\rangle) \\ &= (a_1 \wedge \neg a_2) \vee ((a_1 \leftrightarrow a_2) \wedge ((a_2 \wedge \neg(a_1 \oplus a_2)) \vee \ldots)) \\ &= a_1 \end{aligned}$$

So $||2 * a > 1 + a||$ only holds for the propositional interpretations where $\mathfrak{I}(a_1) = 1$. Indeed, the corresponding Diophantine interpretations with $\mathcal{D}_{\mathfrak{I}}(a) = 2$ or $\mathcal{D}_{\mathfrak{I}}(a) = 3$ are the only ones satisfying the constraint $2 * a > 1 + a$ (if we are restricted to $\mathcal{D}(a) \in \{0, \ldots, 3\}$). Finally, we define $||.||$ on non-atomic constraints:

$$||\neg\alpha|| = \neg||\alpha|| \quad \text{and} \quad ||\alpha \circ \beta|| = ||\alpha|| \circ ||\beta|| \text{ for all} \circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\} \tag{13}$$

By Thm. 6, our encoding defined in (7), (8), (10), (11), (12), (13) is correct.

Theorem 6 (Correctness of Encoding Diophantine Constraints). *For every $\alpha \in \mathcal{C}$ and every propositional interpretation $\mathfrak{I}$, we have $\mathcal{D}_{\mathfrak{I}}(\alpha) = \mathfrak{I}(||\alpha||)$.*

So to determine the satisfiability of a Diophantine constraint α by a Diophantine interpretation with numbers from $\{0, \ldots, 2^k - 1\}$, we now encode α as a propositional formula $||\alpha||$ and then use a SAT solver to find a model $\mathfrak{I}$ of $||\alpha||$. Thm. 7 shows that the size of our encoding is polynomial.

Theorem 7 (Size of Encoding). *Let $\alpha \in \mathcal{C}$ such that every number in α is $\leq 2^k - 1$. Then the size of $||\alpha||$ is in $\mathcal{O}(|\alpha|^2 * k^2)$, where $|\alpha|$ is the size of α.*

4 Polynomials with Negative Constant

Now we regard polynomials $f_{\mathcal{P}ol}$ which may have a negative constant coefficient (i.e., in (2) one may have $a_0 < 0$). All other coefficients still have to be natural numbers. As demonstrated by the tools TTT [17] and AProVE [14] in the termination competitions, such polynomials (in connection with the DP method) are very helpful in practice. We show how to extend our approach in order to use SAT solvers also for such polynomial interpretations.

As in [3, Ex. 4.28], we replace the rules (v) and (vi) of our TRS by

$$\mathsf{bits}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))).$$

Instead of (viii) and (ix) we get the DPs $\mathsf{bits}^\sharp(\mathsf{s}(x)) \to \mathsf{half}^\sharp(\mathsf{s}(x))$ and $\mathsf{bits}^\sharp(\mathsf{s}(x)) \to \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x)))$. Now there is no polynomial interpretation with non-negative coefficients where the DPs are strictly and the rules are weakly decreasing.

Thus, we use a polynomial interpretation $\mathcal{P}ol_3$ with $\mathsf{half}_{\mathcal{P}ol_3} = x_1 - 1$. However, if one extends such interpretations to terms naively, then terms could be mapped to negative numbers and thus, the resulting order would not be well founded. Hence, [17] proposed the following modification in the definition of $[.]$: $[x] = x$ for all variables x and $[f(t_1, \ldots, t_n)] = \max(f_{\mathcal{P}ol}([t_1], \ldots, [t_n]), 0)$. So if $\mathsf{s}_{\mathcal{P}ol_3} = x_1 + 1$, then $[\mathsf{s}(\mathsf{half}(x))]_{\mathcal{P}ol_3} = \max(\max(x - 1, 0) + 1, 0)$. Now one can again replace inequalities $u \succ v$ (resp. $u \succsim v$) by $[u] > [v]$ (resp. $[u] \geq [v]$).

We are interested in *abstract* polynomial interpretations with variable coefficients. To find suitable values for the coefficients, up to now inequalities like $[u] > [v]$ were transformed into Diophantine constraints by building $\alpha_{[u]-[v]>0}$ etc., cf. (4) and (5). Here, we simply required all coefficients of the polynomial $[u] - [v]$ to be non-negative resp. positive. However, now $[u] - [v]$ contains "max" (i.e., it is no longer a polynomial). Thus, it is unclear how to transform $[u] > [v]$ into a satisfiability problem of a Diophantine constraint.

To solve this problem, let us first regard *concrete* polynomial interpretations (where the coefficients are actual numbers). Here, the occurrences of "max" in inequalities $[u] > [v]$ could be eliminated by case analyses. But to increase efficiency, [17] presented an alternative approach to transform inequalities like $[u] > [v]$ into ordinary polynomial inequalities without "max". The idea is to define an under-approximation $[.]^{left}$ and an over-approximation $[.]^{right}$ which do not contain "max" anymore. Then instead of $[u] > [v]$ one requires $[u]^{left} > [v]^{right}$.

Definition 8 ($[.]^{left}$ and $[.]^{right}$ for Concrete Interpretations [17]). *For every polynomial p we denote its constant part by $con(p)$ and the non-constant part $p - con(p)$ by $ncon(p)$. For any concrete polynomial interpretation $\mathcal{P}ol$ and any term t, we define the polynomials $[t]^{left}_{\mathcal{P}ol}$ and $[t]^{right}_{\mathcal{P}ol}$ as follows:*[7]

[7] If $\mathcal{P}ol$ is clear from the context we again omit the subscript "$\mathcal{P}ol$".

$$[t]^{left} = \begin{cases} t & \text{if } t \text{ is a variable} \\ 0 & \text{if } t = f(t_1,\ldots,t_n),\ ncon(p_1) = 0,\ \text{and } 0 > con(p_1) \\ p_1 & \text{if } t = f(t_1,\ldots,t_n),\ \text{otherwise} \end{cases}$$

$$[t]^{right} = \begin{cases} t & \text{if } t \text{ is a variable} \\ ncon(p_2) & \text{if } t = f(t_1,\ldots,t_n) \text{ and } 0 > con(p_2) \\ p_2 & \text{if } t = f(t_1,\ldots,t_n),\ \text{otherwise} \end{cases}$$

where $p_1 = f_{\mathcal{P}ol}([t_1]^{left},\ldots,[t_n]^{left})$ *and* $p_2 = f_{\mathcal{P}ol}([t_1]^{right},\ldots,[t_n]^{right})$.

As shown in [17], we have $[t]^{left} \leq [t] \leq [t]^{right}$ for all terms t. Moreover, if the polynomial interpretation has no negative constants, then we have $[t]^{left} = [t] = [t]^{right}$. For the polynomial interpretation with $\mathsf{half}_{\mathcal{P}ol_3} = x_1 - 1$, we obtain

$$[\mathsf{half}(x)]^{left}_{\mathcal{P}ol_3} = x - 1 \qquad [\mathsf{half}(x)]_{\mathcal{P}ol_3} = \max(x-1, 0) \qquad [\mathsf{half}(x)]^{right}_{\mathcal{P}ol_3} = x \quad \text{(xiii)}$$

The reason is that for both $i \in \{1,2\}$, we have $p_i = \mathsf{half}_{\mathcal{P}ol_3}(x) = x - 1$ and thus $ncon(p_i) = x$ and $con(p_i) = -1$. If $\mathcal{P}ol_3$ is defined like our previous interpretation $\mathcal{P}ol_1$ on all remaining function symbols except half, then we obtain $[u]^{left} > [v]^{right}$ for all DPs $u \to v$ and $[\ell]^{left} \geq [r]^{right}$ for all rules $\ell \to r$. Thus, the termination of our modified example can now easily be shown.

The disadvantage of Def. 8 is that one can only compute $[t]^{left}$ and $[t]^{right}$ for concrete polynomial interpretations.[8] However, if one wants to find the coefficients of the polynomial interpretations automatically, then it would be better to start with *abstract* polynomial interpretations again where the coefficients a_i in (2) are left open (i.e., they are *variable coefficients*).

For example, we would use an abstract interpretation $\mathcal{P}ol_2$ with $\mathsf{half}_{\mathcal{P}ol_2} = a\,x_1 + \boldsymbol{b}$. Here, a may only be instantiated by natural numbers, whereas we denote Diophantine variables like $\boldsymbol{b}$ that may be instantiated by integers in **bold** face. However, to compute $[\mathsf{half}(x)]^{left}_{\mathcal{P}ol_2}$ and $[\mathsf{half}(x)]^{right}_{\mathcal{P}ol_2}$ we would have to decide whether $ncon(p_i) = a\,x$ and $con(p_i) = \boldsymbol{b}$ are equal to resp. less than 0. This of course depends on the instantiation of the variable coefficients a and $\boldsymbol{b}$.

Therefore, we now modify Def. 8 to make it suitable for abstract polynomial interpretations. The idea is to introduce new variables $\boldsymbol{b}_t^{\boldsymbol{left}}$ and b_t^{right} for any term t and to create Diophantine constraints α_t^{left} and α_t^{right} which guarantee that $\boldsymbol{b}_t^{\boldsymbol{left}}$ and b_t^{right} are instantiated correctly. To this end, we express the conditions $ncon(p_1) = 0$ and $0 > con(p_i)$ from Def. 8 as Diophantine constraints.

Definition 9 ($[.]^{left}$ and $[.]^{right}$ for Abstract Interpretations). *For any abstract polynomial interpretation $\mathcal{P}ol$ and any term t, we define:*

- *If t is a variable, then* $[t]^{left} = t$, $[t]^{right} = t$, $\alpha_t^{left} = true$, *and* $\alpha_t^{right} = true$.
- *If* $t = f(t_1,\ldots,t_n)$, *then*[9] $[t]^{left} = ncon(p_1) + \boldsymbol{b}_t^{\boldsymbol{left}}$, $[t]^{right} = ncon(p_2) + b_t^{right}$,

[8] Thus, current implementations for negative polynomials like TTT and AProVE simply *test* several choices for the coefficients. More sophisticated algorithms for *systematically finding* coefficients like [8] only work for non-negative coefficients.

[9] Note that according to Def. 8, $[t]^{left} = ncon(p_1)$ if $ncon(p_1) = 0$ and $0 > con(p_1)$.

$$\begin{aligned}\alpha_t^{left} \;=\; & \alpha_{t_1}^{left} \wedge \ldots \wedge \alpha_{t_n}^{left} \wedge (\quad \alpha_{ncon(p_1)=0} \wedge 0 > con(p_1) \;\rightarrow \boldsymbol{b_t^{left}} = 0) \\ & \wedge (\neg(\alpha_{ncon(p_1)=0} \wedge 0 > con(p_1)) \rightarrow \boldsymbol{b_t^{left}} = con(p_1))\end{aligned}$$

$$\begin{aligned}\alpha_t^{right} = \; & \alpha_{t_1}^{right} \wedge \ldots \wedge \alpha_{t_n}^{right} \wedge (\quad 0 > con(p_2) \;\rightarrow b_t^{right} = 0) \\ & \wedge (\neg(0 > con(p_2)) \rightarrow b_t^{right} = con(p_2))\end{aligned}$$

Here, p_1 and p_2 are defined as in Def. 8 and $\alpha_{ncon(p_i)=0}$ is defined as in (6).

For $\mathsf{half}_{\mathcal{P}ol_2} = a\,x_1 + \boldsymbol{b}$ and $t = \mathsf{half}(x)$, we have $ncon(p_i) = a\,x$, $con(p_i) = \boldsymbol{b}$,

$$[\mathsf{half}(x)]^{left}_{\mathcal{P}ol_2} = a\,x + \boldsymbol{b_t^{left}} \qquad \text{and} \qquad [\mathsf{half}(x)]^{right}_{\mathcal{P}ol_2} = a\,x + b_t^{right} \tag{xiv}$$

$$\alpha_t^{left} = ((a = 0 \wedge 0 > \boldsymbol{b}) \rightarrow \boldsymbol{b_t^{left}} = 0) \;\wedge\; (\neg(a = 0 \wedge 0 > \boldsymbol{b}) \rightarrow \boldsymbol{b_t^{left}} = \boldsymbol{b}) \tag{xv}$$

$$\alpha_t^{right} = ((0 > \boldsymbol{b}) \rightarrow b_t^{right} = 0) \;\wedge\; (\neg(0 > \boldsymbol{b}) \rightarrow b_t^{right} = \boldsymbol{b}) \tag{xvi}$$

Thm. 10 shows that Def. 9 extends Def. 8 to abstract interpretations correctly.

Theorem 10 (Correspondence of Def. 8 and 9). *Let $\mathcal{D}$ be a Diophantine interpretation (which may also map* **bold** *variables to integers). Let $\mathcal{P}ol$ be an abstract polynomial interpretation, and let t be a term. Then $\mathcal{D}(\alpha_t^{left}) = 1$ implies $\mathcal{D}([t]^{left}_{\mathcal{P}ol}) = [t]^{left}_{\mathcal{D}(\mathcal{P}ol)}$ and $\mathcal{D}(\alpha_t^{right}) = 1$ implies $\mathcal{D}([t]^{right}_{\mathcal{P}ol}) = [t]^{right}_{\mathcal{D}(\mathcal{P}ol)}$.*

For example, let $\mathcal{D}$ be an interpretation which turns the abstract polynomial interpretation $\mathcal{P}ol_2$ into the concrete interpretation $\mathcal{P}ol_3$. Thus, we have $\mathcal{D}(a) = 1$ and $\mathcal{D}(\boldsymbol{b}) = -1$ and indeed, $\mathcal{D}(\mathsf{half}_{\mathcal{P}ol_2}) = \mathcal{D}(a\,x_1 + \boldsymbol{b}) = x_1 - 1 = \mathsf{half}_{\mathcal{P}ol_3}$. To satisfy the Diophantine constraints α_t^{left} and α_t^{right} in (xv) and (xvi), we must have $\mathcal{D}(\boldsymbol{b_t^{left}}) = -1$ and $\mathcal{D}(b_t^{right}) = 0$. Then by (xiii) and (xiv) we indeed obtain

$$\begin{aligned}\mathcal{D}([\mathsf{half}(x)]^{left}_{\mathcal{P}ol_2}) &= \mathcal{D}(ax + \boldsymbol{b_t^{left}}) \;= x - 1 = [\mathsf{half}(x)]^{left}_{\mathcal{P}ol_3} \\ \mathcal{D}([\mathsf{half}(x)]^{right}_{\mathcal{P}ol_2}) &= \mathcal{D}(ax + b_t^{right}) = x \qquad\; = [\mathsf{half}(x)]^{right}_{\mathcal{P}ol_3}\end{aligned}$$

So we generate Diophantine constraints containing **bold** variables like $\boldsymbol{b}$ and $\boldsymbol{b_t^{left}}$ which may be instantiated by *integers*. However, our encoding to propositional formulas in Sect. 3 only handles instantiations with *natural* numbers. Therefore, we now show how to remove **bold** variables from constraints α.

In the encoding $||\alpha||$, we restricted ourselves to interpretations $\mathcal{D}$ where for all (non-**bold**) variables a we have $\mathcal{D}(a) \in \{0, \ldots, 2^k - 1\}$ for some fixed $k \geq 1$. Now one has to fix an additional number $n \geq 0$ and for all **bold** variables $\boldsymbol{a}$, we restrict ourselves to $\mathcal{D}(\boldsymbol{a}) \in \{-n, \ldots, 2^k - 1 - n\}$. Hence, to encode a Diophantine constraint α with **bold** variables, we first replace every **bold** variable $\boldsymbol{a}$ in α by "$a - n$" for a fresh (non-**bold**) variable a. Then (after removing subtractions), one can again use our encoding $||.||$ from Sect. 3.

To summarize, the procedure from the end of Sect. 2 to transform a termination problem into a satisfiability problem is now modified as follows:

1. Transform the termination problem to inequalities $u \succ v$ or $u \succsim v$, cf. (1).
2. Fix an abstract polynomial interpretation and transform the inequalities into $[u]^{left} - [v]^{right} > 0$ or $[u]^{left} - [v]^{right} \geq 0$, respectively. Add the conjunction of all corresponding constraints α_u^{left} and α_v^{right}.

3. Replace $[u]^{left} - [v]^{right} \; {}_{(}\!\geq_{)} \; 0$ by $\alpha_{[u]^{left}-[v]^{right} \; {}_{(}\!\geq_{)} \; 0}$.
4. Fix a number $n \geq 0$ and replace all Diophantine variables $\boldsymbol{a}$ that may be instantiated by integers by "$a - n$" for a fresh variable a.
5. Remove "$\geq$" and subtractions from the obtained constraint and check its satisfiability using SAT solving as in Sect. 3.

5 Implementation, Experiments, and Conclusion

We implemented our new SAT-based approach for polynomial interpretations in the termination prover AProVE [14]. We used the MiniSAT solver [9] and to convert formulas to CNF, we applied SAT4J's [21] implementation of Tseitin's algorithm [24]. For efficiency, our implementation uses several optimizations:

(a) Simplification: In addition to standard simplifications for Diophantine constraints and for propositional formulas, we developed a new graph-based approach to detect possible simplifications of Diophantine constraints quickly. We build a graph whose nodes consist of all occurring Diophantine variables and of all possible values they can take (e.g., $\{0, ..., 2^k - 1\}$). An edge from a node n_1 to n_2 denotes that $\mathcal{D}(n_1) \geq \mathcal{D}(n_2)$ for any Diophantine model $\mathcal{D}$ of the given Diophantine constraint. This graph is constructed and maintained while performing the other simplifications. Whenever there is a non-trivial strongly connected component (SCC) in the graph, we can deduce that all its nodes must take the same value under any Diophantine model. If there is more than one number in the SCC, then the Diophantine constraint is not satisfiable. If there is one number in the SCC, we instantiate all Diophantine variables in the SCC by that number. If the SCC only consists of Diophantine variables, we choose an arbitrary one and replace all other variables in the SCC by the chosen one.

(b) Sharing: We use sharing for common subexpressions, both on the level of Diophantine constraints and on the level of propositional formulas.

(c) Tracking maximum values: By taking into account that Diophantine variables are only instantiated by values from a certain set (e.g., $\{0, ..., 2^k - 1\}$), one can keep track of the maximum possible values for all polynomials occurring in the Diophantine constraint. This can help to improve the conversion from Diophantine constraints to tuples of propositional formulas. The reason is that we can detect cases where the most significant bits are equivalent to 0.

As an example, suppose that all Diophantine variables can take values from $\{0, ..., 3\}$ and that consequently, the conversion $||.||$ transforms Diophantine variables into tuples of two propositional variables (i.e., $k = 2$). Note that by definition, $B^*(\langle\varphi_1, ..., \varphi_n\rangle, \langle\psi_1, ..., \psi_m\rangle)$ is always a tuple of length $n+m$, if $m \geq 2$. So if $a, b, c \in \mathcal{A}$, then $||a||$ and $||b||$ have length 2, $||a*b||$ has length 4, and $||a*b*c||$ has length 6. However, if one takes the ranges of the coefficients into account, then one can determine that $a*b*c$ has at most the value $3*3*3 = 27$. Thus, only 5 bits are needed for $||a*b*c||$, i.e., the most significant bit of $||a*b*c||$ is always equivalent to 0. Therefore, it can be omitted (i.e., one should delete the leftmost formula in the 6-tuple $||a*b*c||$, resulting in a 5-tuple).

This optimization is particularly helpful when using other ranges than $\{0, ..., 2^k - 1\}$ (e.g., when using $\{0, 1, 2\}$ instead of $\{0, 1, 2, 3\}$). Then we have to introduce subformulas that prohibit certain values for the Diophantine variables, but this usually pays off due to the reduced search space.

To evaluate our new SAT-based implementation of polynomial interpretations (AProVE-SAT), we compared it with the non-SAT-based implementations in the termination tools AProVE 1.2 and TTT [17]. In addition, we experimented with a version of AProVE which uses the Diophantine solver of the CiME-tool [7] (AProVE-CiME). The implementations in AProVE 1.2 and AProVE-CiME solve Diophantine constraints by a specialized finite domain constraint satisfaction procedure [8], while TTT uses a "generate-and-test" approach instead. Moreover, we considered a variant AProVE-CLP which applies the *constraint logic programming* engine of SICStus Prolog to find polynomial interpretations.

Finally, we also implemented a variant AProVE-PB which uses the *pseudo-boolean solver* Pueblo [23]. Here, instead of encoding Diophantine constraints to propositional formulas, we adapted the encoding $||.||$ from Sect. 3 in order to yield *pseudo-boolean constraints*: For Diophantine variables a over $\{0, \ldots, 2^k - 1\}$ we now define $||a|| = 2^{k-1}\, a_1 + \ldots + 2\, a_{k-1} + a_k$, and we define $||n|| = n$ for $n \in \mathbb{N}$ and $||p \circ q|| = ||p|| \circ ||q||$ for polynomials $p, q \in \mathcal{P}$ and $\circ \in \{+, *\}$. Afterwards, the resulting constraints are linearized.

We tested the six tools on all 865 TRSs from the *Termination Problem Data Base 3.2*.[10] This is the collection of examples used in the *International Competition of Termination Tools 2006*. For our experiments, the tools were run on an AMD Athlon 64 at 2.2 GHz. To measure the effect of the different implementations for polynomial interpretations, we configured all tools to use only a basic version of the DP method and no other termination technique.[11]

For each example, we imposed a time limit of 60 seconds (corresponding to the way tools are evaluated in the annual competition) or of 10 minutes, indicated by "*Limit*" in the following table. The columns "*Yes*" and "*TO*" show the number of TRSs for which proving termination with the given configuration succeeds or times out. Finally, "*Time*" gives the total time in seconds needed for analyzing all 865 examples. The column "*Range*" specifies the range of the coefficients of polynomials (i.e., if the "*Range*" is n, then we only searched for coefficients from $\{0, ..., n\}$). The column "*Degree*" gives the degree of the polynomials. If the "*Degree*" is 1, then we used linear polynomials and "sm" means that we used simple-mixed[12] polynomials (these are not available in TTT).

The comparison of the SAT-based configurations AProVE-SAT and AProVE-PB with the non-SAT-based configurations shows that the provers based on SAT solving with our proposed encoding are faster by orders of magnitude. This holds in particular if one considers a higher time limit or polynomials with higher coefficients or degrees (which are needed to increase the number of "*Yes*"-results,

[10] The data base is available from `http://www.lri.fr/~marche/tpdb/`

[11] Such a configuration was not possible for other tools beside AProVE, TTT, and CiME.

[12] A non-unary polynomial (with $n > 1$ in (2)) is *simple-mixed* if we have $e_{ij} \leq 1$ for all its exponents. A unary polynomial is simple-mixed if it has the form $a + b\, x_1 + c\, x_1^2$.

i.e., to increase the power of automated termination proving). Note that for *Degree* = 1, there are no timeouts in the configuration AProVE-SAT, whereas the non-SAT-based configurations have many timeouts. Due to the increased efficiency, the number of examples where termination can be proved within the time limit is considerably higher in the SAT-based configurations. To indicate the size of the SAT problems obtained, the largest resulting propositional formula contained almost 3.5 million variables and more than 12 million clauses. Comparing the SAT-based configurations AProVE-SAT and AProVE-PB shows that the approach of converting termination problems to propositional formulas is currently preferable to the related approach of converting them to pseudo-boolean constraints.

			AProVE-SAT			AProVE-PB			AProVE 1.2		
Limit	Range	Degree	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time
60s	1	1	421	0	45.5	421	0	61.6	421	1	151.8
60s	2	1	431	0	91.8	431	0	158.5	414	48	3633.2
60s	3	1	434	0	118.6	434	1	222.1	408	81	5793.2
60s	3	sm	440	51	5585.9	427	82	7280.3	404	171	11608.1
10m	1	1	421	0	45.5	421	0	61.6	421	1	691.8
10m	2	1	431	0	91.8	431	0	158.5	418	41	27888.4
10m	3	1	434	0	118.6	434	0	689.6	415	53	38286.4

			AProVE-CLP			AProVE-CiME			TTT		
Limit	Range	Degree	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time
60s	1	1	420	16	1357.8	408	1	168.3	326	32	2568.5
60s	2	1	420	37	3558.3	408	43	3201.0	335	83	5677.6
60s	3	1	407	91	6459.5	402	67	5324.1	338	110	7426.9
60s	3	sm	367	145	10357.4	361	147	10107.7			
10m	1	1	421	11	7852.2	408	0	332.7	328	16	14007.8
10m	2	1	423	25	18795.6	412	33	22190.4	337	68	45046.6
10m	3	1	420	51	41493.8	407	46	33873.6	340	91	61209.2

We also ran experiments with higher ranges but it turned out that they are rarely needed. For *Degree* = 1 and *Limit* = 10 minutes, a range of 6 would increase the number of "*Yes*"-results from 434 to 436 while the runtime increases from 118.6 to 748.1 seconds. Even if one uses a range of 63, the number of "*Yes*"-results does not increase further, but the runtime goes up to 56235.5 seconds.

The next table shows the effect of our optimizations(with linear polynomials and a 60 seconds time limit). While AProVE-SAT uses all optimizations (a) - (c), we also give the results obtained if one omits any one of these optimizations. The table demonstrates that each optimization has a considerable positive effect, especially if one uses higher ranges for the coefficients.

	AProVE-SAT			no optimization (a)			no optimization (b)			no optimization (c)		
Range	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time
1	421	0	45.5	421	0	56.6	421	0	49.7	421	0	50.1
2	431	0	91.8	431	0	107.5	431	0	93.9	431	0	114.7
3	434	0	118.6	434	1	159.4	434	0	202.8	434	0	138.7

The last table demonstrates the use of SAT solving for negative linear polynomials with a time limit of 60 seconds. If the "*Range*" is n, then now the constant coefficient may take values from $\{-n, ..., n\}$. Again, the SAT-based configuration is much faster and substantially more powerful than the non-SAT-based ones. Compared to the results for non-negative

	AProVE-SAT			AProVE 1.2			TTT		
Range	Yes	TO	Time	Yes	TO	Time	Yes	TO	Time
1	440	0	98.0	441	22	1863.7	341	106	7307.3
2	479	1	305.4	460	126	8918.3	360	181	12337.3
3	483	4	1092.4	434	221	15570.9	361	247	16927.7

polynomials, a few timeouts occur for larger ranges, but negative polynomials increase the power significantly whereas the runtimes only increase moderately. In future work, we will extend our SAT encoding in order to deal also with polynomials where other (non-constant) coefficients can be negative [17].

As mentioned in Sect. 1, the SAT-based implementation of polynomial interpretations was used by AProVE in the *International Competition of Termination Tools 2006*. Here, AProVE was configured to use several other termination techniques in addition to polynomial interpretations. Due to the speed of our new SAT-based approach, AProVE could try polynomial interpretations (also with higher ranges) as one of the first termination techniques. In case of failure, there was still enough time to try other termination techniques afterwards. With a time limit of 60 seconds for each example, AProVE could prove termination of 633 TRSs and thereby it was the winner of the competition.

To summarize, automated termination analysis is a field where SAT solving has turned out to be extremely useful. At the same time, this field also poses new challenges for SAT solving, since for higher ranges and higher degrees of the polynomials, one sometimes obtains SAT problems which are hard for current SAT solvers.[13] To experiment with our implementation, for further details on our experiments (also with other SAT solvers), and for all proofs please see [10].

Acknowledgments. We thank Daniel Le Berre for helpful comments.

References

1. E. Annov, M. Codish, J. Giesl, P. Schneider-Kamp, and R. Thiemann. A SAT-based implementation for RPO termination. In *Short Papers of LPAR '06*, 2006.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133-178, 2000.
3. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, 2001.
4. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge, 1998.
5. M. Codish, V. Lagoon, and P. Stuckey. Solving partial order constraints for LPO termination. In *Proc. RTA '06*, LNCS 4098, p. 4-18, 2006.
6. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Proc. LPAR '06*, LNAI 4246, p. 30-44, 2006.
7. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME. `http://cime.lri.fr`.
8. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *J. Aut. Reason.*, 34(4):325-363, 2005.
9. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT '03*, LNCS 2919, p. 502-518, 2004.
10. Empirical evaluation of "SAT solving for termination analysis with polynomial interpretations". `http://aprove.informatik.rwth-aachen.de/eval/SATPOLO`.
11. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In *Proc. IJCAR '06*, LNAI 4130, p. 574-588, 2006.

[13] We have therefore submitted some of these problems to the SAT competition 2007.

12. J. Giesl, R. Thiemann, P. Schneider-Kamp. The DP framework: Combining Techniques for Automated Termination Proofs. *LPAR'04*, LNAI 3452, p.301-331, 2005.
13. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated termination analysis for Haskell: From term rewriting to programming languages. In *Proc. RTA '06*, LNCS 4098, p. 297-312, 2006.
14. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the DP framework. *Proc. IJCAR '06*, LNAI 4130, p. 281-286, 2006.
15. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3): 155-203, 2006.
16. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172-199, 2005.
17. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474-511, 2007.
18. D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proc. RTA '06*, LNCS 4098, p. 328-342, 2006.
19. H. Hong and D. Jakuš. Testing positiveness of polynomials. *JAR*, 21(1):23-38, 1998.
20. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
21. D. Le Berre et al. SAT4J satisfiability library for Java. `http://www.sat4j.org`.
22. P. Schneider-Kamp, J. Giesl, A. Serebrenik, R. Thiemann. Automated termination analysis for logic programs by term rewriting. In *Proc. LOPSTR '06*, LNCS, 2007.
23. H. M. Sheini and K. A. Sakallah. Pueblo: A hybrid pseudo-boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:61-96, 2006.
24. G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, p. 115-125, 1968.
25. H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for argument filterings. In *Proc. SOFSEM '07*, LNCS 4362, p. 579-590, 2007.
26. H. Zankl and A. Middeldorp. KBO as a satisfaction problem. *Proc. WST'06*, 2006.

Fault Localization and Correction with QBF*

Stefan Staber and Roderick Bloem

Graz University of Technology

Abstract. In this paper, we study the use of QBF solvers for fault localization and correction of sequential circuits. Given a violated specification, we compute whether the circuit can be repaired by evaluating a sequence of quantified Boolean formulas. If a repair exists, it can be extracted from a certificate for another quantified Boolean formula. Because it only finds components when a repair is possible, this approach is more precise than a satisfiability-based approach that we have developed earlier. We demonstrate this in an experimental evaluation.

1 Introduction

One of the major applications of SAT solvers is verification of finite-state systems. When a system is faulty, a SAT-based model checker will emit a counterexample, i.e., a sequence of inputs that leads to a failure. Even with a counterexample, it is hard to find and correct the fault, and debugging typically takes a significant amount of the design time.

In previous work, we have addressed the problem of fault localization using a SAT solver. To localize a fault, we see a failure as a contradiction between the behavior described by the circuit and the behavior required by the specification for the given input sequence. A *fault candidate* is a component (e.g., a gate) such that the contradiction disappears when we lift the restriction imposed by the component. Conceptually, we do this by removing the component's clauses from the SAT instance. If this resolves the contradiction then there is a replacement for the component that fixes the circuit for the given counterexample.

This approach has two shortcomings. First, the approach is not very *specific*. It is restricted to one (or a few) counterexamples and thus may find fault candidates for which there is no replacement that is correct for *all* input sequences. Second, this approach does not suggest a replacement for the component, even if one exists.

In this paper we show how quantified Boolean formula (QBF) solvers can be used to address these problems. We use a game-based approach to construct a QBF formula that states that a component is a fault candidate if and only if it can be replaced by a function that is correct for all inputs. If we use a QBF solver that emits a *certificate*, we can use it to construct a replacement for the component. We show how the QBF problem is formulated and how a repair can be built from certificates. We show how to use QBF to solve safety and Büchi games. Although the formulation for safety games closely monitors what is done for Binary Decision Diagrams, the formulation for Büchi games

* This work was supported in part by the European Union under contract 507219 (PROSYD).

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 355–368, 2007.

is novel and has a linear number of quantifier alternations instead of a quadratic number. The results of the QBF solver can be used to construct repairs if the specification is given as a deterministic Büchi automaton. This class includes all safety properties. The formulation of the QBF problem is much like the one for bounded model checking (BMC) [BCCZ99] with a different quantifier structure. Although the approach is complete, limits on time or memory use may force us to limit the search depth. If we limit the search depth, the approach still has much higher specificity that the SAT-based approach.

In [SFBD06], we described how SAT-based fault localization can be used to find a fault on the source (Register Transfer) level. The same applies for the QBF approach, but for simplicity this paper will focus on faults at the gate level. In [GSB06], we have shown the applicability of SAT-based fault localization to C programs. QBF could also be used in that setting, provided we limit the length of the counterexamples. Finally, in [JGB05, SJB05] we presented an approach to localization and correction that is based on BDDs. In contrast, this paper considers quantified Boolean formulas.

In [ASV$^+$05] a QBF solver is used to debug hardware. The approach uses a QBF solver to perform fault localization in a single pass for all given test vectors. These technique was applied on the gate level (on different hierarchical levels) and under the assumption that correct output values for counterexamples are given.

The flow of the paper is as follows. In Section 2 we state the necessary theory. In Section 3 we summarize the use of sat-solvers for fault localization and discuss its shortcomings. In Section 4, we discuss the use of QBF solvers to localize and correct faults. We present preliminary experimental results (on localization, not on repair) in Section 5 and conlude in Section 6.

2 Preliminaries

2.1 Quantified Boolean Formulas

A quantified Boolean formula in prenex conjunctive normal form is defined as a conjunction of clauses with a quantifier prefix. The prefix is a sequence of alternating existentially and universally quantified variables. More formally, a quantified Boolean formula is a formula $F = QB$, where the prefix $Q = q_1 v_1 q_2 v_2 \dots q_n v_n$ is a sequence of quantifiers $q_i \in \{\exists, \forall\}$ and variables v_i, and B is a conjunction of clauses in CNF. Every variable in B is quantified once in the prefix Q. For example:

$$F = \forall a \forall b \exists c.(\neg a \vee \neg b \vee c) \wedge (a \vee b \vee \neg c)$$

A variable v is *existential* if it is existentially quantified, and *universal* otherwise. Variable v is *dominated* by a variable v' if in the prefix v' is quantified before v. It is possible to convert every formula to prenex conjunctive normal form and in the paper we will use arbitrary nestings of Boolean operators and quantifiers. Egly et al. [ETW02] show how to convert formulas efficiently such that the number of quantifier alternations is small. We will use capital letters for sets of variables and quantification over such sets is an abbreviation for a sequence of quantifications, one for each variable in the set. A formula F with free variables V is denoted $F(V)$. If $|V| = |V'|$, we will write $F(V')$

to denote syntactic substitution of the unbound occurrences of the variables in V by variables in V'.

Suppose $e_1, \ldots, e_m$ is the set of existential variables in a quantified Boolean function. Given an existential variable e_i let U_i be the set of universal variables that dominate e_i. A *determining function* for an existential variable e_i is a function that maps a valuation of the variables U_i to a value for e_i. A *certificate* for a QBF is a set of determining functions, one for each existential variable, that proves the truth of the formula [Ben05a]. A *partial certificate* is a set of determining functions for the variables $e_1, \ldots, e_k$ with $k \leq m$ such that can be extended to a certificate by adding functions for $e_{k+1}, \ldots, e_m$. The tool `sKizzo` supports the construction of certificates for quantified Boolean programs. Partial certificates are currently not supported by QBF solvers, but are likely easier to construct than full certificates.

Typically, there exists more than one certificate for the satisfiability of a QBF, since in some valuations the value of an existential variable is not relevant. Certificates can be represented as Boolean functions, truth tables, or BDDs. As an example, one certificate for F is $c = a \wedge b$, another one is $c = a \vee b$.

2.2 Circuits and Games

A sequential circuit consists of a set of gates, a set of signals connecting the gates, and a set of flipflops with a given initial state. Signals are connected to at most one output of a gate. If a signal is not connected to the output of a gate, it is an input of the circuit. We will assume that all cycles in a circuit contain a flipflop, we will assume a single global clock, and we are not concerned with timing issues.

A circuit can be represented as a *symbolic transition structure* (V, I, s_0, T), where V, the set of variables has a one to one correspondence to the flipflops, I is the set of input signals, s_0 is a Boolean formula over the variables which is only satisfied by the initial state, and T, the transition function, is a Boolean formula over V, I, and V'. Intuitively, T specifies a relation over $V \times I \times V$ that corresponds to the combinatorial logic of the circuit. We can build T as follows: suppose that the circuit contains n gates, numbered 1 through n and that gate i is represented by a Boolean function φ_i in terms of the input and the output signals. Then $T = \bigwedge_i \varphi_i$. We can assume that the φ_i are given in CNF, which implies that T is also in CNF.[1] We have $n = 2^{|V|}$ states, and the *diameter* d of the circuit is maximum length of any shortest path from the initial state to another reachable state.

A game is an extension of a transition structure: A *(deterministic) game* is a tuple (V, I, C, s_0, T), together with a specification ψ, where V, I, and s_0 are as before, C is a set of *system choice variables*, and T is now a formula over V, I, C, and V' that defines a function from $V \times I \times C$ to V. One way to view a game [AH99] is as a partially implemented system: the inputs are provided by an antagonistic environment that wants the system to violate its specifications, and the system choices are provided by a protagonist that wants the system to fulfill the specifications. The natural question

[1] We will conveniently ignore the fact that T contains a free variable for every signal. When T is used in a quantified Boolean formula, these variables should be existentially quantified in the appropriate place.

is thus whether the protagonist can match, move by move, any input sequence by a sequence of choices so that ψ is fulfilled. If this is the case, the game is *won*. A *play* is an infinite sequence $\pi = s_0, i_0, c_0, s_1, i_1, c_1, \ldots$ of states, inputs, and choices that adheres to the transition relation. A *strategy* maps a prefix of a play to a system choice. A strategy fixes a set of plays, one for each input sequence. The strategy is *winning* if each play thus obtained satisfies the specification. A strategy is *memoryless* if it only depends on the last state and the last input in the prefix. Given a memoryless strategy, we can *restrict* the game to the strategy by fixing the choice for every combination of a state and an input, thus obtaining a circuit. A strategy is *finite state* if it can be implemented using a finite memory of past inputs, using a finite state machine.

In this paper, we consider safety and Büchi games, i.e. games in which the winning condition is either given as a set of states that must be avoided (*losing states*), or by a set of states that must be visited infinitely often. Both types of games have memoryless strategies [Tho95]. In fact, for these games there is either a winning memoryless strategy for the protagonist or a winning memoryless strategy for the antagonist. When a specification is instead given as a safety property, we can compose the game with the automaton for the safety property and use the algorithm for safety games to either decide that the game is lost or to obtain a memoryless winning strategy. The memoryless strategy can in turn be used to obtain a finite-state strategy for the original game. (The memory is provided by the automaton for the safety property.) This technique is known as a *game reduction* and can also be applied if the specification is given as a separate deterministic Büchi automaton.

2.3 Fault Localization and Correction

Suppose we are given a circuit and a specification ψ and suppose the circuit does not satisfy the specification. Assume that the fault can be fixed by replacing a single gate.

We are looking for a simple replacement for the faulty gate. Thus, we allow only combinatorial corrections. The combinatorial replacement for a gate is a function in terms of the current state given by the flipflops and the current input values. We do not allow sequential corrections because this would require the introduction of additional flipflops. Such a correction may alter the circuit significantly and could be hard to understand for the debugging engineer.

In order to find the fault, we turn the circuit into a game. In this game, the protagonist decides which gate is incorrect and how this gate should behave. As before, assume that we have n gates. The game is defined as $G = (V, I, \{ab^1, \ldots, ab^n, S\}, s_0, T)$, where V is the set of flipflops, I is the set of inputs and s_0 is the initial state, as before. The system choice variables consist of a set $\mathrm{AB} = \{\mathrm{ab}_1, \ldots, \mathrm{ab}_n\}$ of n *abnormal* signals and a variable S. Intuitively, ab_i means that gate i is incorrect and S is a new signal, a Boolean variable that determines the replacement behavior of the gate. We have $T = \bigwedge_i \varphi_i'$, where

$$\varphi_i' = (\neg\, \mathrm{ab}_i \wedge \varphi_i) \vee (\mathrm{ab}_i \wedge S),$$

where φ_i is the formula describing the behavior of gate i, as above.

Let ψ' be the conjunction of the specification ψ, the requirement that the choice for all ab_i is constant, and the requirement that exactly one of the ab_i is set.

If ψ is an invariant, the resulting game is a safety game. If it is a set of states to be visited infinitely often, G is a Büchi game.

Theorem 1. *[JGB05] Suppose G is a safety game. There is a winning strategy that sets ab_i to 1 if and only if there is a combinational function f in terms of inputs and states such that the circuit is correct when gate i is replaced by f.*

Suppose G is a Büchi game. Then there is a combinational replacement for gate i iff there is a winning strategy that sets ab_i to 1.

If ψ is a safety property or a property expressed as a deterministic Büchi automaton, we need to add a monitor to the circuit. In this case it should be noted that the repair may depend on the state of the monitor [JGB05].

It may be doubted that a replacement of a single gate would suffice to fix a typical design error. However, this approach can be lifted to the Register Transfer (RT) level, by correcting entire expressions. Since this approach works on the same level as the designer, it is more likely to find usable results. The technical approach to finding and fixing faults at the RTL level is not much different from the gate level, so we do not describe it here. Similarly, the approach is easily extended to localization of multiple faults. See [JGB05, SJB05, GSB06, SFBD06].

3 SAT-Based Fault Localization

In this section we summarize the SAT-based approach for fault localization presented in [SFBD06] and discuss its shortcomings. The SAT-based approach builds a propositional formula such that any satisfying assignment of the formula corresponds to fault candidate. Suppose we are given a faulty sequential circuit, a specification ψ, and a finite counterexample ξ with length k.

To construct the propositional formula we extend the circuit to a game as discussed in Section 2.3. As in BMC, we unroll the transition function T with length k. We concatenate the unrolled transition function with the specification ψ, a formula ζ_1 that states that most one of the AB variables is one, and the counterexample ξ. By adding ξ to the formula, we fix the inputs of the system. In game terms, we fix the moves of the antagonistic environment. Now, we have to find system choices from the initial state to state k in order to fulfill ψ. We obtain the formula: $\eta = s_0 \wedge \psi \wedge \xi \wedge \zeta_1 \wedge \bigwedge_{i\in 0\ldots k-1} T(V_i, I_i, AB \cup \{G_i\}, V_{i+1})$. We use a SAT solver to decide the satisfiability of η. If the SAT solver finds a satisfying assignment for η, one signal ab_i will be set to one. The corresponding gate i is a fault candidate. Typically, there is more than one assignment to AB for which the η is satisfiable. Each such assignment corresponds to a fault candidate. We can obtain all fault candidates by excluding candidates found previously using blocking clauses.

3.1 Example

We illustrate the SAT-based approach on a simple arbiter example shown in Figure 1.

The arbiter has two request lines $r1, r2$ and two acknowledge lines a_1, a_2. Latch C1 is high if a request on r_1 is not acknowledged, latch C2 stores open requests for r_2.

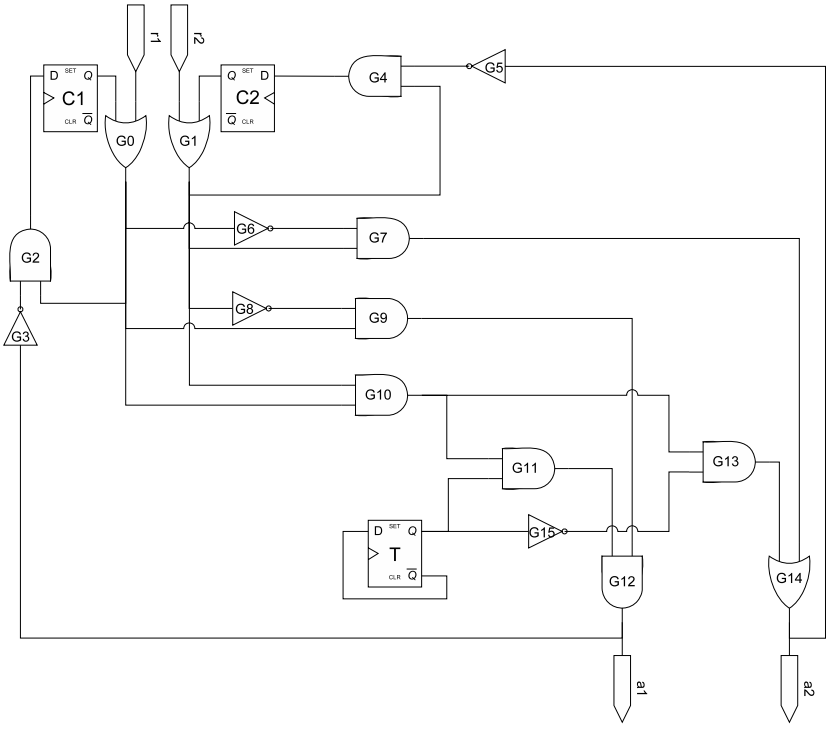


Fig. 1. Simple Arbiter

Latch T decides which request line has priority if both are high. The initial values of the latches are zero.

The specification states that the arbiter has to guarantee that a request is acknowledged in the same or the next clock cycle and at most one of the acknowledge lines may be high at any clock cycle. In LTL, the specification reads $\mathsf{G}\,\neg(a1 \wedge a2) \wedge \mathsf{G}(r1 \rightarrow (a1 \vee \mathsf{X}\,a1)) \wedge \mathsf{G}(r2 \rightarrow (a2 \vee \mathsf{X}\,a2))$.

The arbiter contains a fault. Gate G12 should be an OR gate. A shortest counterexample to the specification is $r_1 = 1, r_2 = 0$ in the first clock cycle and $r_1 = 0, r_2 = 1$ in the second cycle. Given this input, the request outputs are both zero in both cycles. When we use this counterexample for fault localization, we obtain the fault candidates G8, G9, G11, and G12. With each of these gates, it is possible to remove the contradiction between the specification and this particular counterexample. For instance, if G11 returns 1 in the initial cycle, the output will be $a1 = 1$, $a2 = 0$ in the first cycle, and $a1 = 0$, $a2 = 1$, which is correct.

3.2 Shortcomings

For the calculation of the fault candidates we use a single counterexample. The reported fault candidates can be used to correct the circuit, but *only* for the used counterexample.

Table 1. Values for Arbiter

	Input		State			Output		Correction
	r_1	r_2	C1	C2	T	a_1	a_2	G12
Clock Cycle 1	1	0	0	0	0	0	0	0
Clock Cycle 2	0	1	1	0	1	0	0	1
Clock Cycle 1	1	0	0	0	0	0	0	1
Clock Cycle 2	1	0	1	0	1	0	0	0

For instance, although we showed a replacement for G11 that works for the input described above, there is no correction for G11 such that the specification holds for all possible inputs. This is easily seen: If input r_2 is always high, G9 is always low and so is G12, regardless of G11. Thus, a request on r_1 cannot be acknowledged. This is an example of a spurious fault location that has no realizable correction.

By considering multiple counterexamples, we can shrink the set of fault locations and, by judiciously choosing the counterexamples, we can also remove G11. However, picking the proper counterexamples can be hard and should not be left to the user.

Apart from this lack of specificity, the approach does not allow us to construct repair. For instance, Table 1 shows the input values, output values and a correction that the SAT solver may provide for gate G12. The input values and the state are the same in the first clock cycle for both counterexamples. However, the suggested correction for counterexample one is 0 but for counterexample two, it is 1. Gate G12 is repairable with an OR-gate, but with the reported valuations we cannot infer a consistent correction.

As an extreme example, the algorithm may find fault candidates even if the given specification is not realizable. Assume we have given a circuit which consists only of a simple inverter, and the specification reads $b1 \Leftrightarrow \mathsf{X}\, a1$. Clearly, this specification is not realizable because the output depends on future input. The shortest counterexample for this specification has length two. E.g., two consecutive 0s on the input produce two consecutive 1s on the output. The SAT-based approach, however, declares the inverter to be a fault candidate. The satisfying assignment sets the output to 0 in the first cycle, which fulfills the specification. Note that with the given formalization, it is impossible conclude that the buffer is not a fault candidate. (See [SFBD06] for a partial solution to this problem.)

4 QBF-Based Fault Location and Correction

In this section we describe an approach for fault localization and correction based on quantified Boolean formulas and certificates. The goal of the QBF approach is to provide increased specificity and correct repair suggestions. In fact, the fault candidates found will be exactly those gates for which a repair exists. We will first describe how to solve a safety game using QBF and then show a simple optimization for our application. Following that, we discuss Büchi games and give an example.

4.1 QBF for Safety Games

As discuss in Section 2, in safety games the specification corresponds to a set of states in which the protagonist must remain. The environment, the antagonist in our game, tries to force a visit to a bad state, where the specification does not hold. We define the predicate $B(V)$ to be the set of bad states.

It is well known that the set of losing states in a safety game can be computed as

$$\mu Y.\, B(V) \vee \exists I \forall C \exists V'.\, T(V, I, C, V') \wedge Y(V'),$$

where μ denotes the least fixed point. Let W be the set of winning states, i.e., the complement of the fixed point. A winning strategy is any function that maps the combination of a state q in W and an input i to a choice c such that $T(q, i, c, q')$ and $q' \in W$. We can mirror the computation of this fixed point as a sequence of quantified Boolean formulas. We define the following formulas, where intuitively Y_i denotes that the environment can force a visit to Y in i steps.

$$\begin{aligned} Y_0(V) &= B(V), \\ Y_i(V) &= B(V) \vee \exists I \forall C \exists V'.T(V, I, C, V') \wedge Y_{i-1}(V'). \end{aligned}$$

We compute Y_i for increasing i until either the initial state is included in Y_i, which means that the game is lost, or until Y_i implies Y_{i-1}, which implies that a fixpoint has been reached and the game is won. In the following algorithm, $Y_i(V)$ is a formula with free variables V. (Thus, Y_i is a syntactic construct and not evaluated to a set). The function QBF_Evaluate is a decision procedure that takes a closed quantified Boolean formula and returns true iff the formula is valid. The algorithm either terminates with the message "lost", or with a formula $W(V)$ that describe the set of won states.

```
i := 0;
Y_i(V) := B(V);
while (true) {
  if QBF_Evaluate(∀V.s_0(V) → Y_i(V)) then print "lost"; stop;
  if QBF_Evaluate(∀V.Y_i(V) → Y_{i-1}(V)) then break;
  i := i + 1;
  Y_i(V) := B(V) ∨ ∃I∀C∃S'.T(V, I, C, V') ∧ Y_{i-1};
}
// Game won, Y_i and Y_{i-1} are equivalent, but latter is smaller.
k := i - 1;
W(V) := ¬Y_k(V);
```

If the game is won, we can extract a winning strategy from W. In this case, the following formula is valid and its certificate maps every combination of a winning state and an input to a choice. The certificate is a winning strategy.

$$Z = \forall V.W(V) \rightarrow \forall I \exists C \exists V'.T(V, I, C, V') \wedge W(V')$$

Note that a partial certificate suffices: we need a certificate for C but not for the existentially quantified variables inside the two occurrences of W.

Note that the maximum number of quantifier alternations in Y_k is $2k + 1$, where k is the value of i upon termination of the algorithm. This corresponds exactly to the number

of quantifier alternations in a BDD-based version of the algorithm. (The BDD-based algorithm would make k *symbolic steps* [BGS06], each computing $\exists I \forall C.T(V, I, C, V') \wedge Y(V')$, where Y is a BDD.) The number of quantifier in Z is $2k + 1$.

Note that much work is repeated evaluating the termination criteria. Thus, an incremental version of QBF_Evaluate may be beneficial. The number of calls to QBF_Evaluate is $2k$ but can be reduced to $O(\log(k))$ by a binary search or to 1 or 2 by choosing i sufficiently large (e.g., equal to the number of states).

4.2 QBF for Repair

Returning to our application of repair, recall that the choice variables consist of a set AB of abnormal predicates and a variable S that states the new behavior. The requirement that the variables AB do not change allows us to pull them out of the quantification, so that Y_i becomes

$$Y_i'(V_0) = \forall \mathrm{AB}\,.\zeta_1 \wedge B(V_0) \vee \exists I_1 \forall S_1 \exists V_1.T(V_0, I_1, \mathrm{AB} \cup \{S_1\}, V_1) \wedge (B(V_1) \vee \ldots \\ \exists I_i \forall S_i \exists V_i.T(V_{i-1}, I_i, \mathrm{AB} \cup \{S_i\}, V_i) \wedge B(V_i)) \ldots),$$

where again, ζ_1 denotes that at most one component is abnormal. Furthermore,

$$Z' = \exists\, \mathrm{AB}\, \forall V.\, \zeta_1 \wedge (W(V) \rightarrow \forall I \exists S \exists V', T(V, I, \mathrm{AB} \cup \{S\}, V') \wedge W(V'))$$

Note that the fault candidate can be derived from a partial certificate for Y_k. The algorithm stops as soon as one fault candidate, say l, has been identified. Other fault candidates can be found by requiring that ab_l is zero and restarting the algorithm.

If the game is won, we obtain a certificate for Z. The certificate consists of:

- A (constant) function f_1 that gives an appropriate value for AB.
- A function f_2 that maps every combination of a state and an input to an appropriate value for G, i.e., an appropriate output for the fault candidate.

When we implement the function f_2 as a combinational circuit, we obtain a replacement for the fault candidate. Synthesis of the function can easily be automated if the certificate is given as a BDD or a truth table.

Note that we may terminate the algorithm before reaching convergence in case the QBF solver runs out of time or memory. In this case, we can still extract a repair suggestion using the certificate for Z'. If we terminate the algorithm after i steps, the suggested repair guarantees that there is no counterexample of i steps or less, but there is no guarantee that a bad state is never reached. The suggestion may, however, give a good hint on how to repair the system.

4.3 QBF for Büchi Games

The approach described thus far work for safety properties. We will now consider liveness. In particular, we will assume that the repair problem is stated as a Büchi game. Using the same techniques as for safety properties, we can handle techniques in which the property is stated as a deterministic Büchi automaton. Such automata can not express all properties, but are sufficiently expressive to state most properties of interest

[Mai00, ALT04, JGB05]. We will first describe how Büchi games are computed and then briefly explain how to use the theory for repair.

Suppose we have a game (V, I, C, s_0, T) and the winning condition requires that any play visit the set F infinitely often. Typically, Büchi games are computed by the evaluating the following nested fixed point. (Where ν denotes a greatest fixed point.)

$$\nu Y.\, \forall I \exists C \exists V'.\, T(V, I, C, V') \wedge \\ \mu Z.\, Y(V') \wedge (F(V') \vee \forall I' \exists C' \exists V''.\, T(V', I', C', V'') \wedge Z(V'')).$$

The total number of iterations of the inner fixed point in this formula is quadratic in the size of the state space [BGS06]. Thus, when evaluating this formula using, e.g., BDDs, the total number of quantifier alternations that is evaluated is quadratic. We can easily mimic this fixpoint as a sequence of quantified Boolean formulas, as we did for safety games. The drawback is that such formulas can have a quadratic number of quantifier alternations. We will present an alternative approach that needs only a linear number of quantifier alternations.

Let us define

$$W_k^0(V_0, \ldots, V_{k-1}) = \forall I_k \exists C_k \exists V_k.\, T(V_{k-1}, I_k, C_k, V_k) \wedge \\ \bigvee_{i \in [0,k]} \bigvee_{j \in [i+1,k]} V_i = V_j \wedge \bigvee_{l \in [i,j]} F(V_l),$$

$$W_k^i(V_0, \ldots, V_{k-i-1}) = \forall I_{k-i} \exists C_{k-i} \exists V_{k-i}.\, T(V_{k-i-1}, I_{k-i}, C_{k-i}, V_{k-i}) \wedge W_k^{i-1}, \text{and}$$

$$W_k(V_0) = W_k^{k-1}(V_0).$$

Intuitively, $s \models W_k(V_0)$ if the protagonist can force the game into an accepting loop within k steps. Similarly, we have

$$L_k^1(V_0, \ldots, V_{k-1}) = \exists I_k \forall C_k \exists V_k.\, T(V_{k-1}, I_k, C_k, V_k) \wedge \\ \bigvee_{i \in [0,k]} \bigvee_{j \in [i+1,k]} V_i = V_j \wedge \bigwedge_{l \in [i,k]} \neg F(V_l),$$

$$L_k^i(V_0, \ldots, V_{k-i-1}) = \exists I_{k-i} \forall C_{k-i} \exists V_{k-i}.\, T(V_{k-i-1}, I_{k-i}, C_{k-i}, V_{k-i}) \wedge L_k^{i-1}, \text{ and}$$

$$L_k(V_0) = L_k^{k-1}(V_0).$$

Intuitively, $s \models L_k(V_0)$ if the antagonist can force the game into a non-accepting loop within k steps.

Suppose the game is won. Then, the protagonist has a winning memoryless strategy. If we restrict the game so that the protagonist's choices adhere to the strategy, the resulting circuit will contain only fair loops (and unreachable loops). Thus, any input sequence of sufficient length will visit a loop containing an accepting state and $s_0 \models W_k$ for some k smaller than or equal to the diameter of the game. Simultaneously, $s_0 \not\models L_k$ for any k: there are no loops without an accepting state. Through a similar argument we can show that if the game is lost, we have $s_0 \models L_k$ for some k but $s_0 \not\models W_k$ for any k. This suggests the following algorithm. (We leave out the construction of the formulas.)

```
i := 0;
while (true) {
  if QBF_Evaluate(∀V.s0(V) → Li(V)) then print "lost"; stop;
  if QBF_Evaluate(∀V.s0(V) → Wi(V)) then break;
  i := i + 1;
}
k := i;
W(V) := Wk(V);
```

The strategy can be extracted from a certificate in the same way as for a safety game: We check if the protagonist can force the game to a state satisfying $R(V) = W(V) \wedge F(V)$. Let $Y_i(S)$ be the formula that denotes that the protagonist can reach $R(V)$ in i steps or less. Let

$$Z = \forall S \forall I \exists C \exists S'.\, T(S, I, C, S') \wedge \bigvee_{i>0} ((Y_i(S) \wedge \neg Y_{i-1}(S) \wedge Y_{i-1}(S')) \vee (F(S) \wedge W(S'))).$$

A certificate for Z is a memoryless strategy, that is, it maps S and I to C. The strategy corresponds to the standard *attractor strategy* for Büchi games: it always moves closer to a winning state in F and from there it moves to an arbitrary winning state. In detail, we have that a state s satisfies $Y_i(S) \wedge \neg Y_{i-1}(S)$ if the protagonist can force a visit to a state in $F \wedge W$ in i steps but not fewer. If that is the case, the protagonist needs to move to a state from which it can reach $F \wedge W$ in at most $i - 1$ steps, which is denoted by $Y_{i-1}(S')$. If the state is in F, we have $F(S)$, and in the next state we must remain in a winning state: $W(S')$.

It is interesting to note that this algorithm is quite different from the standard BDD-based one that uses a quadratic number of steps. (More efficient algorithms are not known, but neither is a lower bound.) In our case it suffices to call the QBF solver $\log n$ times, or once or twice, if we choose i sufficiently large. Each call considers a formula with at most n quantifier alternations.[2]

As with safety games, when applying this approach to repair, we can move the quantification of the AB variables to the outer position. Since we again obtain a memoryless strategy, the conversion of a strategy to a repair is as with safety games.

4.4 Example

We repeat the arbiter example of section 3 with our implementation of the QBF approach for safety games. In Section 3, we showed that the SAT-based approach provided four fault candidates for the counterexample.

For the QBF-approach we added two additional flipflops to the circuit. The flipflops act as monitor and keep track of requests that were not acknowledged. With help of the monitor we can translate the original safety specification into an invariant. The approach returns gate G12 as the only possible fault location.

[2] As stated, the length of the formulas is quadratic. We can reduce this to linear by checking only if the last state has occurred previously, and by allowing sharing of subformulas. The drawback is that the number of quantifier alternations is no longer limited by the diameter of the game, but by the size of its state space.

The QBF solver delivers a correction for G12. The certificate contains a BDD representing the function

$$G12 = C1 \wedge (C2 \wedge T \wedge (r_1 \vee r_2) \vee \neg C2 \wedge (T \vee r_1 \wedge \neg r_2)) \vee (\neg C1 \wedge T \wedge r_1).$$

We can simplify this function because we know that following conditions must hold in the circuit: $C1 \rightarrow \neg C2, T \rightarrow \neg C2, \neg T \rightarrow \neg C1$. Considering this constraints, we obtain the function $G12 = T \wedge (C1 \vee r_1)$. Since gate G0 is $C1 \vee r_1$ this is equivalent to $G12 = T \wedge G0$. This is a correct repair (and simpler than the original implementation): it produces an $a1$ only when T is true and there is a request either now or in the last tick. An $r2$ is handled when T is low by the remaining logic.

The example shows that the QBF approach overcomes the disadvantages of the SAT-based approach. It is exact and provides only fault locations that can be corrected. Furthermore, the certificate contains a combinatorial correction for the fault candidate. The provided correction can be easily translated into circuitry.

Naturally, we prefer simple corrections for the faulty gate. Since a certificate represents only one possible valuation of the variables, it is not guaranteed that we obtain the simplest possible correction for the faulty gate. In future work we would like to investigate how we can provide corrections as simple as possible.

5 Experiments

In this section we present preliminary experimental results. In the experiments we focus on localization only and we compute the fault candidates for a fixed depth k, k being the length of a counterexample provided by BMC. For the experiments we used benchmarks provided with the VIS model checker [B$^+$96] written in Verilog on RT level. We manually introduced a bug in the examples (on the RT level). The experiments were executed on a Pentium IV with 2,8GHz and 3GB of RAM running Linux.

We used the `sKizzo` QBF solver [Ben05b]. `sKizzo` is able to dump the inference log of a QBF to a file. The certifier program `ozziKs` applies a reconstruction to the log and is able to dump a certificate that is based on BDDs. As far as we know, `sKizzo` is the only QBF solver that is able to generate certificates (We have not yet evaluated whether the proofs that yQuaffle produces [YM05] can be used for the same purpose.)

Table 2 contains the obtained experimental results. We are comparing the results of the SAT-based approach with the QBF approach. Col. one is the name of the circuit. Col. two and three show the number of gates and registers in the design respectively. Col. four shows the number of RT-level components in the design. Col. five shows the length of the counterexample obtained by BMC. Col. six and seven show the number of components in a static slice. The static slice is computed backwards from the output signals. All the components that have an influence on the signals in the property, are part of the static slice. For debugging, only components in the static slice should be considered, so this gives an upper bound on the number of components we should find. For the SAT results in col. eight and nine we used the approach described in [SFBD06]. In col. ten and eleven the number of fault locations for the QBF approach are given and

Table 2. Experimental results. M...memory out; T...time out; a)...we were able to compute only one fault location, subsequently memory out.

Circuit				BMC	Slice		SAT		QBF		time	
Name	Gates	Registers	#cmp	k	#cmp	%	#cmp	%	#cmp	%	sKizzo	2clsQ
b01_e1	98	7	40	5	32	80	5	13	2	6	8:16	0:20
b02_e1	46	4	20	5	20	100	5	25	5	25	0:13	0:03
b09_e1	398	28	33	21	22	67	6	18	1	3	1:20:13	T
b10_e1	318	20	61	7	53	87	10	16	-	-	M	T
b11_e1	770	31	44	6	39	89	9	20	5	11	3:10:11	T
b13_e1	505	53	96	5	72	75	3	3	a)	a)	0:10	1:08
VsaR_e1	2956	154	56	15	50	89	8	14	-	-	M	T

in the last two col. we give the runtimes for finding one fault location with `sKizzo` and 2clsQ [SB06] respectively. The results for the SAT-based approach are significantly better than the static slice. Results for the QBF approach are mixed. In three of the examples we were able to improve the results compared to the SAT-based approach and in one example the number of fault locations could not reduced. In example b13 we were able to compute one fault location before the QBF solver ran out of memory. For two examples we could not provide any fault locations. Note that we have not limited the possible fault locations in the QBF formulas by using the result from the SAT solver or the static slice. This would likely speed up the QBF solver.

6 Conclusions

We have studied the use of QBF solvers to locate and correct faults in finite state systems. Although our exposition has focused on single faults on the gate level, it can easily be extended to multiple faults and the implementation works on the RT level. Since repair corresponds to computing a game, we have shown an algorithm for computing safety games and a novel algorithm for computing Büchi games which has a linear number of quantifier alternations. Furthermore, we have shown how certificates can be used to derive strategies and thus repairs.

Our experimental results are still preliminary. They show that the QBF approach is slow, but more precise than a competing SAT-based approach. We have also seen that repairs can be extracted. The repair approach performs many very similar computations. It would benefit from an incremental approach like the one for SAT. Likewise, we would like to try to find simple repairs. It would be beneficial if more general certificates could be obtained, i.e., relations instead of functions. It should be noted that `sKizzo` derives certificates that contain more information than needed and that partial certificates would suffice.

Acknowledgments. We would like to thank Armin Biere and Marco Benedetti for fruitful discussions and the latter for a pre-release of `ozziKs`.

References

[AH99] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[ALT04] R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Transactions on Computational Logic*, 5(1):1–25, January 2004.

[ASV+05] M. Ali, S. Safarpour, A. Veneris, M. Abadir, and R. Drechsler. Post-verification debugging of hierarchical designs. In *Proc. IEEE International Conference on Computer Aided Design (ICCAD 2005)*, pages 871–876, San Jose, California, USA, 2005.

[B+96] R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eighth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Fifth International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, pages 193–207, March 1999. LNCS 1579.

[Ben05a] M. Benedetti. Extracting certificates from quantified Boolean formulas. In *International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 47–53, 2005.

[Ben05b] M. Benedetti. sKizzo: a suite to evaluate and certify QBFs. In *Proc. of 20th International Conference on Automated Deduction (CADE05)*, 2005. LNCS 3632.

[BGS06] R. Bloem, H. N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. *Formal Methods in System Design*, 28:37–56, 2006.

[ETW02] U. Egly, H. Tompits, and S. Woltran. On quantifier shifting for quantified boolean formulas. In *Proceedings of the SAT-02 Workshop on Theory and Applications of Quantified Boolean Formulas (QBF-02)*, pages 48–61, 2002.

[GSB06] A. Griesmayer, S. Staber, and R. Bloem. Automated fault localization for C programs. In *Workshop on Verification and Debugging (V&D'06)*, 2006. To Appear.

[JGB05] B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In K. Etessami and S. K. Rajamani, editors, *17th Conference on Computer Aided Verification (CAV'05)*, pages 226–238. Springer-Verlag, 2005. LNCS 3576.

[Mai00] M. Maidl. The common fragment of CTL and LTL. In *Proc. 41th Annual Symposium on Foundations of Computer Science*, pages 643–652, 2000.

[SB06] H. Samulowitz and F Bacchus. Binary clause reasoning in QBF. In *Proc. 9th Intern. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, 2006.

[SFBD06] S. Staber, G. Fey, R. Bloem, and R. Drechsler. Automatic fault localization for property checking. In *Second Haifa Verification Conference*, 2006.

[SJB05] S. Staber, B. Jobstmann, and R. Bloem. Finding and fixing faults. In D. Borrione and W. Paul, editors, *13th Conference on Correct Hardware Design and Verification Methods (CHARME '05)*, pages 35–49. Springer-Verlag, 2005. LNCS 3725.

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. In *12th Annual Symposium on Theoretical Aspects of Computer Science*, pages 1–13. 1995. LNCS 900.

[YM05] Y. Yu and S. Malik. Validating the result of a quantified Boolean formula (QBF) solver: Theory and practice. In *Asia and South Pacific Design Automation Conference (ASPDAC'05)*, pages 1047–1051, 2005.

Sensor Deployment for Failure Diagnosis in Networked Aerial Robots: A Satisfiability-Based Approach

Fadi A. Aloul[1] and Nagaragan Kandasamy[2]

[1] Department of Computer Engineering, American University of Sharjah, UAE
faloul@aus.edu
[2] Department of Electrical and Computer Engineering, Drexel University, USA
kandasamy@ece.drexel.edu

Abstract. Unmanned aerial vehicles (UAVs) represent an important class of networked robotic applications that must be both highly dependable and autonomous. This paper addresses sensor deployment problems for distributed failure diagnosis in such networks where multiple vehicles must agree on the fault status of another UAV. Sensor placement is formulated using an integer linear programming (ILP) approach and solved using Boolean satisfiability (SAT)-based ILP solvers as well as generic ILP solvers. Our results indicate that the proposed models are tractable for medium-sized UAV networks.

Keywords: 0-1 ILP, SAT, UAV networks, fault diagnosis, distributed systems.

1 Introduction

Unmanned aerial vehicles (UAVs) represent an important class of robotic applications for distributed sensing and control. A collection of vehicles must perform a shared task while coordinating the required inter-vehicle actions using wireless communication. Examples include remote sensing, surveillance and patrol, and data collection over areas dangerous to human intervention. Such UAV networks have significant cost constraints. However, they must be both highly dependable and largely autonomous, requiring only high-level guidance from ground controllers.

Sensing and surveillance applications require that UAV node maintain a tight spatial formation or physical topology, including specified inter-node distances. In a typical decentralized formation-control scheme, each node receives information from neighboring nodes such as their position and velocity, and uses this data for local control aimed at maintaining its position within the topology [6]. Therefore, correct and timely information flow between nodes is critical to maintaining a stable topology.

To maintain the specified topology of a UAV network comprising nodes $N_1,...,N_q$, each N_j must communicate some critical information such as its position and velocity to neighboring nodes. Hardware (software) failures may, however, cause the node to transmit erroneous values. Though *physical redundancy* in the form of replicated sensors and processors can mask such node failures, it also adds to N_j's

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 369 – 376, 2007.

cost, weight, and power consumption. A low-cost alternative is failure diagnosis using *analytical redundancy* [8] where other nodes in the topology use their local sensors and an appropriate mathematical model to estimate the values sent by N_j, and compare discrepancies between the actual and estimated values.

This paper addresses sensor deployment problems for distributed failure diagnosis in wireless UAV networks where multiple nodes must agree on the fault status of another node. We assume that a node N_i in this topology requires a testing configuration-a set of sensors-to monitor N_j for example, if N_i has a GPS sensor, and additionally, a 3D laser range finder, it can, using these sensors and an appropriate mathematical model, independently estimate N_j's position. Several choices of testing configurations are typically available for N_i, differing from each other in their monitoring range, detection capabilities, and cost. (Another possible testing configuration on N_i may comprise a 2D laser range finder and an omni-directional camera.) Also, the sensors themselves may have varying operating distances. Clearly, long-range sensors can monitor multiple nodes, and at greater distances. However, the use of such expensive sensors may substantially increase the overall system cost. On the other hand, if only short-range sensors are used, effective diagnosis may only be achieved with a large number of such sensors. Therefore, efficient sensor selection and placement strategies are needed to minimize system cost while achieving the desired level of diagnosability.

Previous research has addressed distributed system diagnosis under the assumption that processing units test each other and exchange the test results to identify failures [2]. Failed units are then removed from future computations. Several variants of this problem have been studied in the literature, including diagnosing transient and intermittent faults [10], probabilistic diagnosis [4], and failure diagnosis in random, sparse, and highly regular topologies [7]. Since explicit tests are typically difficult to obtain in practice, various comparison-based approaches have also been proposed, where tasks are duplicated on multiple units and their results compared to identify faulty ones [3]. A good survey of prior diagnosis-related research is presented in [2]. The above papers, however, don't address the sensor selection and placement problems for failure diagnosis in wireless networks.

The authors of [5] present a method to identify faulty processors in ad hoc wireless networks via a comparison-based diagnosis model. They present algorithms for both fixed and time-varying network topologies, and show that diagnosis efficiency is significantly reduced when the topology changes with time. As before, sensor selection and placement problems are not addressed.

The sensor placement problem is related to both the alarm and guard placement problems [12, 13]. In [12], alarms are placed on the nodes of a failure propagation graph such that one failed node is uniquely and efficiently identified. A fault propagates along this graph activating one or more alarms and the diagnosis algorithm finds the node responsible for causing them. The guard placement problem can be informally stated as that of determining the minimum number of guards, each having a certain monitoring range, to cover the interior of an art gallery, represented as a polygon [13].

This paper uses an integer linear programming (ILP) approach to solve sensor deployment problems for distributed failure diagnosis in UAV networks. We specifically target popular UAV formations such as mesh, diamond, and circular topologies [15, 16] , and provide exact solutions for topologies up to 40 nodes, representative of topology sizes assumed by researchers while developing formation control algorithms [15, 17].

The proposed method aims to minimize both the testing and communication costs associated with identifying a bounded number of faulty UAV nodes. (In a typical wireless network, it is desirable to minimize the transmitting range of individual nodes to reduce power consumption and network interference.) Assuming an upper bound f on the number of node failures, we formulate and solve ILP models for the following optimization problem: Given a topology comprising q empty slots and an equal number of UAV nodes, each having a specific testing and communication configuration, allocate nodes to slots such that *system diagnosability*, in terms of the number of diagnosed nodes, is maximized. The above is termed the *MaxD* problem.

The model are solved using two different 0-1 ILP (SAT-based and generic-based) solvers [1, 9] and their performance is compared. Our experiments indicate that these models are tractable for topologies up to forty nodes.

The rest of this paper is organized as follows. Section 2 discusses some modeling assumptions and the distributed diagnosis approach. We develop ILP models for the *MaxD* problem in Section 3 and solve them in Section 4. We conclude this paper in Section 5.

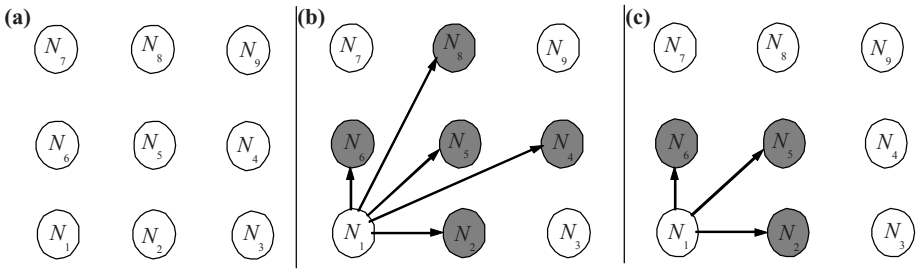


Fig. 1. (a) A grid topology of UAVs. The testing edges induced on the other nodes when N_1 chooses (b) a testing configuration T_1 and (c) a shorter-range testing configuration T_2 .

2 Preliminaries

This section describes the assumed system model and discusses the distributed diagnosis approach. The combinatorial nature of the sensor selection and placement problems of interest is briefly outlined.

2.1 System Model

We assume a distributed system where UAV nodes communicate with each other over a wireless network having limited bandwidth and must maintain the specified physical topology. Fig. 1(a) shows a grid topology for UAVs. High-level controllers

coordinate with other nodes of interest to maintain the topology while feedback-control loops regulate local dynamics on each node.

A node N_i's position within a topology is given in the (x_i, y_i, z_i) dimensions and the distance between nodes N_i and N_j is

$$D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \quad (1)$$

When N_i has a choice of testing configurations, we let T_{ik} denote the k^{th} such configuration with testing range $range(T_{ik})$ and cost a_{ik}; if $range(T_{ik}) \geq D_{ij}$, then N_i can test (or monitor) N_j using configuration T_{ik}. Similarly, if C_{il} denotes the l^{th} communication configuration on N_i having cost b_{il}, then node N_i can transmit messages to N_j if $range(C_{il}) \geq D_{ij}$. (Also, whenever the context is clear, we will refer to the k^{th} testing and l^{th} communication configuration on a node simply as T_k and C_l, respectively.)

As noted in Section 1, controllers on each N_j must communicate some critical information such as its position and velocity to neighboring nodes to maintain the desired topology. We assume that N_j may suffer operational failures including permanent and transient ones, thereby transmitting erroneous (sensor) information to its neighbors. Therefore, N_j must be diagnosed and removed from participating in future formation-control computations.

2.2 Distributed Diagnosis

Distributed diagnosis in a topology such as Fig. 1(a) requires that multiple testing nodes agree on the fault status of a testee node N_j. This is achieved using a 2-phase approach as follows. During phase 1, each testing node independently evaluates the information transmitted by N_j. These local decisions are then consolidated via a suitable agreement algorithm during phase 2 to obtain a global view of N_j's status. Similar 2-phase diagnosis schemes have been previously proposed to identify faulty processors [14].

We assume an analytical redundancy-based checking scheme that is executed locally on node N_i to evaluate the information sent by N_j. Node N_i uses its onboard testing configuration and an appropriate mathematical model to independently estimate N_j's sensor values. These estimates are compared to the actual values sent by N_j to generate a residue or error. During phase 2, N_i exchanges the locally generated residue with other testing nodes within communication range. Since multiple testers may employ both design and data diversity, i.e., use various testing configurations and/or models to estimate the same values, these residues may differ slightly from each other, and yet be correct. Therefore, each tester obtains a voted residue value using an approximate agreement algorithm, and evaluates it against an a *priori* defined threshold to diagnose N_j. If all testers perceive N_j's failure uniformly, then

a suitable agreement algorithm is the median voter which selects the middle value from an odd number of residues by eliminating those residue pairs differing by the greatest amount [11]. At the end of phase 2, all fault-free nodes correctly identify N_j's status.

Assuming an upper bound f on the number of node failures in the topology, we need at least $2f+1$ tester nodes to diagnose another node. The distributed approach described above also tolerates failures during the diagnosis process itself and increases confidence in the corresponding decisions. Finally, to reduce the cost of diagnosis, not all sensors on N_j are diagnosed. A few critical sensors are typically selected and checkers implemented to diagnose them.

Returning to Fig. 1, a testing configuration selected for N_i induces corresponding testing edges on neighboring nodes where $N_i \rightarrow N_j$ indicates that N_i can monitor N_j. Fig. 1(b) shows the edges generated when N_1 chooses a testing configuration T_1. Fig. 1(c), on the other hand, shows the case where a testing configuration T_2 with a shorter range is used. We also assume "line-of-sight" testing, i.e., there must be an uninterrupted path between the testee and tester nodes. Therefore, in Fig. 1, node N_1 cannot test N_3, N_7, and N_9, since they are not in the line-of-sight.

3 Problem Formulation

Given a topology with q empty slots and an equal number of nodes, each with a specific testing and communication configuration, allocate nodes to slots such that system diagnosability, in terms of the number of diagnosed nodes, is maximized. We assume an upper bound f on the number of node failures. We define the following decision variables.

$x_{ij} = 1$ if N_i occupies slot j; 0 otherwise

$m_{ij} = 1$ if node placed in slot i can moniter the node in j; 0 otherwise

$d_i = 1$ if the node placed in slot i is diagnosable; 0 otherwise

$p_{ij} = 1$ if a node N_i can communicate with N_j; 0 otherwise

We maximize the cost function

$$\sum_{i=1}^{q} d_i \tag{2}$$

subject to the following constraints. A node N_i must be allocated to exactly one slot.

$$\sum_{i=j}^{q} x_{ij} = 1 \quad \forall i \tag{3}$$

Conversely, each slot j must have exactly one node allocated to it.

$$\sum_{j=1}^{q} x_{ij} = 1 \quad \forall j \tag{4}$$

Let s_{ij} denote the set of nodes, when placed in slot i, can monitor slot j; node $N_k \in s_{ij}$ if it has a testing configuration T_{lk} such that $range(T_{lk}) \geq D_{ij}$. Constraint (5) sets the decision variable m_{ij} to indicate if a chosen node-to-slot allocation enables slot i to test slot j, and constraint (6) ensures that a node allocated to slot j is monitored by at least $2f + 1$ other nodes.

$$\sum_{N_k \in s_{ij}} x_{ki} - m_{ij} \geq 0 \quad \forall i, \forall j, i \neq j \tag{5}$$

$$\sum_{i=1}^{q} m_{ij} \geq 2f + 1 \quad \forall j, i \neq j \tag{6}$$

Constraint (7) sets the decision variable p_{ij} indicating if a chosen node-to-slot allocation enables slot i to transmit to slot j. Let s_{ij} now denote the set of nodes, when placed in slot i, have the transmission range to reach slot j; node $N_k \in s_{ij}$ if its communication configuration C_{lk} is such that $range(C_{lk}) \geq D_{ij}$.

$$\sum_{N_k \in s_{ij}} x_{ki} - p_{ij} \geq 0 \quad \forall i, \forall j, i \neq j \tag{7}$$

Constraints (8), (9), and (10) select exactly $2f + 1$ slots to diagnose the node placed in slot j. Note that the node placed in slot j must transmit its sensor values to every member of the selected subset; otherwise it is not diagnosable. For example, if under some node-to-slot allocation, slot j cannot transmit its sensor values to a slot i chosen to monitor it, i.e., $z_{ij} = 1$ and $p_{ij} = 0$, then clearly d_j must be 0 to satisfy constraint (10).

$$m_{ij} - z_{ij} \geq 0 \quad \forall j, \forall i, i \neq j \tag{8}$$

$$\sum_{i=1}^{q} z_{ij} = 2f + 1 \quad \forall j, i \neq j \tag{9}$$

$$d_i + z_{ij} - p_{ij} \leq 1 \tag{10}$$

Finally, for each slot j, the $2f + 1$ slots chosen to diagnose it must be able to exchange the test results amongst themselves and reach an agreement during phase 2 of the diagnosis process. These slots must be fully connected or else the node in slot j cannot be diagnosed. For example, consider a pair of slots i and k chosen to diagnose slot j, i.e., $z_{ij} = z_{kj} = 1$. However, if slots i and k cannot exchange their test results, i.e., if $p_{ik} = 0$ or $p_{ki} = 0$, then clearly d_j must be zero to satisfy both constraints (11) and (12).

$$d_j + z_{ij} + z_{kj} - p_{ik} \leq 2 \tag{11}$$

$$d_j + z_{ij} + z_{kj} - p_{ki} \leq 2 \quad \forall i, j, i \neq j, k \neq (i \vee j) \tag{12}$$

4 Performance Evaluation

We solve the ILP model developed in Section 3 using the generic-based ILP solver CPLEX [9] and the SAT-based 0-1 ILP solver PBS ver. 4 [1]. The CPLEX and PBS solvers were executed on an Intel Xeon 3 GHz machine with 4 GB RAM. The results presented in this section assume *grid* topologies, though the models are directly applicable to other important formations such as circles and diamonds.

Both CPLEX and PBS were used to solve the *MaxD* model for different topology sizes. For each experiment, we generated a grid topology comprising q empty slots. Assuming an equal number of nodes, a specific testing and communication configuration was pre-selected for each node such that the distribution of configurations to nodes was uniform. The time-out periods for the CPLEX and PBS solvers were set to 10,000 seconds.

Table 1 summarizes the results obtained by CPLEX and PBS, in terms of the number of diagnosable nodes, for $f = 1, 2$. Optimal results are shown in boldface in the figures. We assume five testing (communication) configurations corresponding to α values of 0, 0.25, 0.5, 0.75, and 1. The results show that PBS outperforms CPLEX in the $f = 1$ case. Both solvers time-out trying to prove the solution optimality in the $f = 2$ case. To summarize, the *MaxD* model appears tractable for medium-size topologies up to 40 nodes.

Table 1. Number of nodes diagnosed under the difference fault models; five testing (communication) configurations corresponding to α = 0, 0.25, 0.5, 0.75, 1 are assumed

Nodes(q)	*f* = 1				*f* = 2			
	PBS		CPLEX		PBS		CPLEX	
	Cost	Time	Cost	Time	Cost	Time	Cost	Time
4x5	**20**	0.26	**20**	164	16	t/o	15	t/o
5x5	**25**	1.7	**25**	237	20	t/o	19	t/o
5x6	**30**	1.59	**30**	2163	24	t/o	24	t/o
6x6	**36**	10.44	33	t/o	28	t/o	0	t/o

5 Conclusions

This paper has addressed the problem of sensor deployment for distributed failure diagnosis in UAV networks. The *MaxD* model allows designers to specify the placement of nodes within a given topology to maximize system diagnosability while incurring no additional testing costs. The ILP model was solved using the generic-based ILP solver CPLEX and SAT-based 0-1 ILP solver PBS, and experimental results indicate that they are tractable for medium-size topologies. For larger topologies, a straightforward (and sub-optimal) solution is to partition the given topology into

portions tractable for the ILP models, and solve the resulting sub-problems in parallel. We will investigate this and other approximation methods in future work.

References

[1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Generic ILP Versus Specialized 0-1 ILP: An Update," *Proc. IEEE/ACM Conf. Computer Aided Design* (ICCAD), 450-457, November 2002.

[2] M. Barborak, M. Malek, and A. Dahbura, "The Consensus Problem in Fault-Tolerant Computing," *ACM Computing Surveys*, 25(2), 171-219, June 1993.

[3] D. Blough and H. Brown, "The Broadcast Comparison Model for On-line Fault Diagnosis in Multicomputer Systems: Theory and Implementation," *IEEE Trans. Comp.*, 48(5), 470-493, May 1999.

[4] D. Blough, G. Sullivan, and G. Masson, "Efficient Diagnosis of Multiprocessor Systems under Probabilistic Models," *IEEE Trans. Computers*, 41(9), 1126-1136, September 1992.

[5] S. Chessa and P. Santi, "Comparison-Based System-Level Fault Diagnosis in Ad-hoc Networks," *Proc. IEEE Symposium Reliable Distributed Systems*, 257-266, 2001.

[6] J. Fax and R. Murray, "Information Flow and Cooperative Control of Vehicle Formations," *IEEE Transactions on Automatic Control*, 49(9), 1465-1476, September 2004.

[7] D. Fussel and S. Rangarajan, "Probabilistic Diagnosis of Multiprocessor Systems with Arbitrary Connectivity," *Proc. IEEE Symposium on Fault-Tolerant Computing*, 560-565, 1989.

[8] J. Gertler, "Fault Detection and Diagnosis in Engineering Systems," *Marcel Dekker*, NewYork, 1998.

[9] ILOG CPLEX, *http://www.ilog.com/products/cplex*

[10] W. Kozlowski and H. Krawczyk, "A Comparison-Based Approach to Multi-Computer System Diagnosis in Hybrid Fault Situations," *IEEE Trans. Computers*, 40(11), 1283-1287, November 1991.

[11] P. Lorczak, A. Caglayan, and D. Eckhardt, "A Theoretical Investigation of Generalized Voters for Redundant Systems," *Proc. IEEE Symposium on Fault-Tolerant Computing*, 444-451, 1989.

[12] N. Rao, "Computational Complexity Issues in Operative Diagnosis of Graph-Based Systems," *IEEE Trans. Computers*, 42(4), 447-457, April 1993.

[13] J. O'Rourke, "Art Gallery Theorems and Algorithms," *Oxford University Press*, Oxford, 1987.

[14] C. J. Walter, P. Lincoln, and N. Suri, "Formally Verified On-Line Diagnosis," *IEEE Trans. Software Engineering*, 23(11), 684-721, November 1997.

[15] J. Fax and R. Murray. "Information Flow and Cooperative Control of Vehicle Formations," *Proc. IFAC World Congress*, July 2002.

[16] A. Pant et al., "Mesh Stability of Unmanned Aerial Vehicle Clusters," *Proc. American Control Conf.*, 2001.

[17] J. Desai, J. Ostrowski, V. Kumar. "Control of Changes in Formation for a Team of Mobile Robots," *Proc. IEEE Conf. Robotics & Automation*, 1556-1561, May 1999.

Inversion Attacks on Secure Hash Functions Using SAT Solvers

Debapratim De[1], Abishek Kumarasubramanian[1], and Ramarathnam Venkatesan[1,2]

[1] Cryptography, Security and Algorithms Research Group, Mircosoft Research India, Bangalore
[2] Cryptography and Anti-Piracy Research Group, Microsoft Research, Redmond
{dde,abikum,venkie}@microsoft.com

Abstract. Inverting a function f at a given point y in its range involves finding any x in the domain such that $f(x) = y$. This is a general problem. We wish to find a heuristic for inverting those functions which satisfy certain statistical properties similar to those of random functions. As an example, we choose popular secure hash functions which are expected to be hard to invert and any successful strategy to do so will be quite useful. This provides an excellent challenge for SAT solvers. We first find the limits of inverting via direct encoding of these functions as SAT: for MD4 this is one round and twelve steps and for MD5 it is one round and ten steps. Then, we show that by adding customized constraints obtained by modifying an earlier attack by Dobbertin, we can invert MD4 up to 2 rounds and 7 steps in < 8 hours.

1 Introduction

Many combinatorial and optimization problems can be encoded as SAT instances and the efficiency of SAT solvers, which has improved considerably over the years, has enabled solving them. Some of them [1][2] perform very well in practical situations. In this paper, we study the inversion problem, where one is given a function F, as a program or an algorithm, and asked to come up with an algorithm $I(y)$ that given y in the range of F, finds some $x = I(y)$ such that $F(x) = y$. Since any NP problem can be formulated appropriately as an inversion problem, one can expect wide applicability, and indeed expect difficulty (in the worst case) in solving them; we here focus on how to use various SAT solver strategies on those problems of practical interest that satisfy some statistical properties. Among them it is natural to study those extreme cases that present considerable challenge to gain some perspective.

Indeed, if F happens to be a secure hash function such as MD4, MD5 and SHA0, it possesses nice statistical properties akin to a random function: however one expects any SAT solver to fail if one encodes it's inversions directly as a SAT problem. We report on the experiments that show what can be achieved with current SAT solvers. The secure hash functions proceed in rounds, typically 3 or 4, and each round having some fixed number (e.g., 16) of steps. It is common to

J. Marques-Silva and K.A. Sakallah (Eds.): SAT 2007, LNCS 4501, pp. 377–382, 2007.

consider the progress of a cryptographic attack method by studying its efficacy on the function restricted to a reduced number of rounds and steps.

Next we ask the following question: can we use some knowledge about the function F at hand? For example, in the secure hash functions we can expect first two conditions below to hold and the third one to a large extent depending on the SAT encoding:

- EQUIDISTRIBUTION: For any y and L, the number of x of length L that map into the same y is approximately $2^{L-|y|}$.
- INPUT EXTENSIBILITY: Even if one fixes some of the m bits of (or imposes some conditions on) x, one expects solutions for $F(x) = y$ to exist if $2^{L-|y|-m}$ is large enough (for example exceeding 1000).
- AUXILIARY VARIABLE EXTENSIBILITY: Similar claim applies the auxiliary variables: assume the encoding of the equation $F(x) = y$ results in a SAT formula $\phi(x_1, ..., x_n, y_1, ..., y_m, a_1, ..., a_T)$ where x_i and y_j are respectively the bits of x and y, and a_i are the auxiliary variables used in the encoding. Then, on fixing some of the bits of (or imposing some statistically equivalent conditions on) a_i (for suitable chosen values of i) ϕ will still be satisfiable, similar to item 2.

These assumptions allow us to pose more constraints on the variables in the intermediate steps and rounds, thereby reducing the search space. Such an approach was implicitly used by Dobbertin to invert MD4 for 2 rounds with an estimate of 2^{32} MD4 computations. A direct SAT encoding of this attack inverted 2 rounds in less than a second. Typically, to keep our work factor manageable, in our experiments we restricted the run time of the SAT solvers to be within a day. Our results on the limits of direct encoding must be viewed in this context.

Next, we present a way of posing extra constraints on the intermediate values during the computation, so as to achieve an inverse of MD4 up to 2 round and 7 steps. We give data on the results along the intermediate steps, and as well as the variable prioritization techniques to help the solver. The properties of equidistribution and extensibility are statistical in nature and are usually enjoyed by a wide variety of practical problems. It is reasonable to expect these to hold in many other applications and our techniques may be reasonably expected to yield results.

1.1 Prior Work

In the past few years, many cryptographic hash functions have been shown to be vulnerable to collision-finding attacks [3][4][5][6]. There has also been a successful attempt [7] in applying SAT solvers to an essential constraint satisfaction phase in these collision-finding attacks. In this paper, we explore the use of SAT solvers in preimage attacks.

2 Hash Functions

Hash functions are very important and useful cryptographic primitives. They are commonly used in signature schemes, time stamping mechanisms, random number generators and many cryptographic protocols.

A hash function maps an input string of arbitrary or almost arbitrary length to one with a fixed length. A cryptographic hash function H mapping a domain D to a range R is required to satisfy the following three properties - (a) *Collision Resistance:* It should be 'hard' to produce distinct $x_1, x_2 \in D$ such that $H(x_1) = H(x_2)$; (b) *Second Preimage Resistance:* For a given $x_1 \in D$, it should be 'hard' to produce a distinct $x_2 \in D$ such that $H(x_1) = H(x_2)$; (c) *Preimage Resistance:* For a given $y \in D$, it should be 'hard' to produce an $x \in D$ such that $H(x) = y$. We call preimage attacks also as inversion attacks.

3 Inversion Attacks on MD4

One may view MD4 as a map from binary strings into $\{0,1\}^{128}$. It first pads, if necessary, the input so that its length becomes a multiple of 512 then applies a compress function iteratively, which maps 512-bit message blocks to 128 bits. Each iteration consists of 3 rounds, each of which is 16 steps long. Typically, attacks focus on the compress function. For details see [8]. Ours is a pre-image attack on the 2 round and 7 step version.

We now present an algorithmic description suitable for our purposes. It uses an array Q of 32-bit values indexed by $[-3, -2, -1, 0, 1, \ldots, 48]$.

Round 1:
  for $i = 1$ to 16 **do**
    $Q[i] = f(Q[i-4], Q[i-3], Q[i-2], Q[i-1], X[i-1])$
  end for

Round 2:
  for $i = 17$ to 32 **do**
    $Q[i] = g(Q[i-4], Q[i-3], Q[i-2], Q[i-1], X[p(i-17)]$
  end for

Round 3:
  for $i = 33$ to 48 **do**
    $Q[i] = h(Q[i-4], Q[i-3], Q[i-2], Q[i-1], X[q(i-33)]$
  end for

Here the permutations p and q and the functions f, g, h are defined in the MD4 standard.

This computation can be encoded as a SAT formula in the following manner. Consider each of $Q[i], i \in \{-3, -2, -1, \ldots, 48\}$, as an array of 32-bit unknowns. Each of the functions f, g, h involve elementary operations, and are encoded as SAT. In order to encode the inversion attack, we set the output bits to the hash value that we want to invert. The 128 bit hash value output after x rounds is in $Q[x], Q[x-1], Q[x-2]$ and $Q[x-3]$.

Let $IV = (Q[-3], Q[0], Q[-1], Q[-2])$ be the initial chaining vector for MD4. The steps of reduced version of the MD4 algorithm for two rounds and seven steps can be described as an iterative computation using Table 1 [10]. In this table, any empty entry means the value of the register is equal to the entry

directly above it. For eg. the entry in "Register A" at step 7 is Q_5. The table describes the general MD4 computation without any constraints. In the attacks we enforce the equation (*) or (#), given later, and thus, for eg., register 'A', 'C', and 'D' will have value K in step 15.

Direct Encoding Results. A direct encoding of 1 round and 12 steps produces a SAT instance with 491520 clauses and 3476 variables. We found the following results from two inversions of the special all zero hash:

0x975c2001 0x71605f00 0x6fa2ecaa 0x5e3135b0 0x02802000 0x6ef98002
0x97fd84ac 0xe1c01d6c 0x450c1900 0x45e1b020 0xb1456cb6 0x299b16af
0x9a7ca6c0 0xb9220116 0x79dd0069 0x25cdfec4

and

0xeec261af 0x911406c1 0x64aa4753 0x224b8cc0 0xa3048188 0xa39ca000
0xe98a4afd 0xc3f19656 0x051600a0 0xc1c01202 0x6b018627 0xb104fef9
0x50cc0480 0xa0e94340 0xa9da0860 0x063fff72

The first instance was run on SATELITE and MINISAT. It compresses to 202407 clauses and 3007 variables, in 75 seconds, which solves in another 12 seconds. The second instance takes 72 seconds to solve on plain MINISAT.

Dobbertin's Attack. Dobbertin's inversion attack [10] on a two round version of MD4 takes a given hash value Q[32], Q[31], Q[30], Q[29] and finds a value for the message in X[0] through X[15]. Given that one expects a given hash value to have $\frac{2^{512}}{2^{128}}$ inverses (assuming the function behaves like a random one), Dobbertin poses the following constraint on intermediate values

```
Q[25] , Q[26] , Q[27] , Q[21] , Q[22] , Q[23] = K  (*)
Q[17] , Q[18] , Q[19] , Q[13] , Q[14] , Q[15] = K
```

This reduces the search space from 2^{128} to 2^{64} involving an a non-linear implicit 32-bit valued function $\mathcal{D}$ in an equation of the form (we call $Q[128]$ as B_0 to match the notation in [10])

$$\mathcal{D}(B_0, K) = 0 \tag{1}$$

This reduces the attack complexity to (2^{32}) steps on an average: one chooses the values for B_0, K (64-bits) at random and check if (1) holds. This extends automatically further to 2 rounds and 3 steps but further extension by direct encoding were not solvable in a few days, possibly because such encoding increases the search space size to 96-bits.

Variable Prioritization. In the DPLL algorithm, the branch variables can be carefully ordered and tailored to make the solver more efficient with respect to the application at hand and extra knowledge. In the case of Dobbertin's attack, the 32-bit variables B_0, K in the equation 1, are prioritized as branching points ahead of every other variable. Between the bits of B_0, K we allow MINISAT's activity heuristics to dictate variable(bit) ordering. This strategy is used in all attacks.

Table 1. MD4 Computation for 2 Rounds and 7 Steps

First Round Of MD4

Input	Register 'A'	Register 'B'	Register 'C'	Register 'D'	Step
	Q_{-3}	Q_0	Q_{-1}	Q_{-2}	
X_0	Q_1				1
X_1				Q_2	2
X_2			Q_3		3
X_3		Q_4			4
X_4	Q_5				5
X_5				Q_6	6
X_6			Q_7		7
X_7		Q_8			8
X_8	Q_9				9
X_9				Q_{10}	10
X_{10}			Q_{11}		11
X_{11}		Q_{12}			12
X_{12}	Q_{13}				13
X_{13}				Q_{14}	14
X_{14}			Q_{15}		15
X_{15}		Q_{16}			16

Second Round Of MD4

Input	Register 'A'	Register 'B'	Register 'C'	Register 'D'	Step
X_0	Q_{17}				17
X_4				Q_{18}	18
X_8			Q_{19}		19
X_{12}		Q_{20}			20
X_1	Q_{21}				21
X_5				Q_{22}	22
X_9			Q_{23}		23
X_{13}		Q_{24}			24
X_2	Q_{25}				25
X_6				Q_{26}	26
X_{10}			Q_{27}		27
X_{14}		Q_{28}			28
X_3	Q_{29}				29
X_7				Q_{30}	30
X_{11}			Q_{31}		31
X_{15}		Q_{32}			32

Third Round Of MD4

Input	Register 'A'	Register 'B'	Register 'C'	Register 'D'	Step
X_0	Q_{17}				33
X_8				Q_{18}	34
X_4			Q_{19}		35
X_{12}		Q_{20}			36
X_2	Q_{21}				37
X_{10}				Q_{22}	38
X_6			Q_{23}		39

Our Attack. The SAT solver enables one to modify the table by posing more implicit constraints. We remove the constraint on $Q[13]$ and set the rest of the variables in the Dobbertin's constraints to 0. The new set of constraints on the hash values of MD4 are to set the values of

```
Q[25] , Q[26] , Q[27] , Q[21] , Q[22] , Q[23] = 0  (#)
Q[17] , Q[18] , Q[19] , Q[14] , Q[15]         = 0
```

Results. The following are examples of inversions that we obtain using the above extension of the Dobbertin's attack. Recall that no inversions for more than 2 rounds and 3 steps were known before. The hash values were chosen in a special form to highlight the fact that the results were not obtained via forward computation of the messages.

MD4 2 ROUNDS 4 STEPS
0xb7759877 0xa57d8667 0xa57d8667 0x87428825 0xa57d8667 0xa57d8667 0xa57d8667 0xb9ca39c5 0xa57d8667 0xa57d8667 0xa57d8667 0x0ac7e6f0 0x6a59a547 0x8f2c86ce 0xa983dbf1 0x554ad05a
hashes to
0x00000001, 0x00000000, 0x00000000, 0x00000000
MD4 2 ROUNDS 5 STEPS
0x9c013f03 0xa57d8667 0xa57d8667 0xdfc0ab4a 0xa57d8667 0xa57d8667 0xa57d8667 0xee416467 0xa57d8667 0xa57d8667 0xa57d8667 0xb114a392 0x4efced8e 0x82581cf9 0x493bb897 0xa7b1272c
hashes to
0x00000000, 0x00000000, 0x00000000, 0x00000001
MD4 2 ROUND 7 STEPS
0x7cd74bbd 0xa57d8667 0xa57d8667 0x8981e841 0xa57d8667 0xa57d8667 0xa57d8667 0x63ba9d30 0xa57d8667 0xa57d8667 0xa57d8667 0x8e67f411 0x044ef497 0x7b17b462 0x7f884714 0x69725052
hashes to
0x11111111, 0x11111111, 0x11111111, 0x11111111

We refer the reader to the full version of the paper for results on preimage attacks on MD5.

4 Conclusions

We presented heuristics for solving inversion problems for fucntions that satisfy certain statistical properties similar to that of random functions. We demostrate that this technique can be used to solve the hard case of inverting a popular secure hash function. We believe this technique may be extended to invert other hard functions, particularly with a better understanding of their internal structure.

References

1. Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. Proc., Design Automation Conference (DAC), June 2001
2. Eén, N., Sorensson, N.: An extensible SAT solver. Proc., International Symposium on the Theory and Applications of Satisfiability and Testing (SAT), 2003
3. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. Advances in Cryptology, EUROCRYPT 2005. Proc., Volume 3494 of LNCS, Springer, 2005.
4. Wang, X., Yu, H.: How to break MD5 and other hash functions. CRYPTO 2005. Proc., Volume 3621 of LNCS, Springer, 2005.
5. Wang, X., Yu, H., Yin, Y. L.: Efficient collision search attacks on SHA-0. CRYPTO 2005. Proc., Volume 3621 of LNCS, Springer, 2005.
6. Wang, X., Yin, Y. L., Yu, H.,: Finding collisions in the full SHA-1. CRYPTO 2005. Proc., Volume 3621 of LNCS, Springer, 2005.
7. Mironov, I., Zhang, L.: Applications of SAT Solvers to Cryptanalysis of Hash Functions. Proc., International Symposium on the Theory and Applications of Satisfiability and Testing (SAT), 2006.
8. Rivest, R. L.: The MD4 message digest algorithm. Advances in Cryptology-CRYPTO 90, Volume 537 of LNCS, pages 303311, Springer, 1991.
9. Rivest, R. L.: The MD5 message digest algorithm. RFC 1321, The Internet Engineering Task Force, 1992.
10. Dobbertin, H.: The MD4 message digest algorithm. Fast Software Encryption: 5th International Workshop, FSE'98, Volume 1372 of LNCS, Springer, 1998.
11. Eén, N., Biere, A.: Effective Preprocessing in SAT Through Variable and Clause Elimination. Proc., International Symposium on the Theory and Applications of Satisfiability and Testing (SAT), 2005.

Author Index

Aloul, Fadi A. 369
Ansótegui, Carlos 10
Argelich, Josep 28
Audemard, Gilles 16

Bacchus, Fahiem 215
Biere, Armin 201
Bloem, Roderick 355
Bonet, María Luisa 10
Bubeck, Uwe 244
Buresh-Oppenheim, Joshua 300

Cimatti, Alessandro 334

Darwiche, Adnan 294
Davis, Martin 1
De, Debapratim 377
Dershowitz, Nachum 287

Een, Niklas 272

Fuhs, Carsten 340

Giesl, Jürgen 340
Glaß, Michael 56
Gomes, Carla P. 100
Griggio, Alberto 334

Hanna, Ziyad 287
Haubelt, Christian 56
Heras, Federico 41
Hertel, Alexander 159
Hertel, Philipp 159
Heule, Marijn 134, 258
Hoffmann, Joerg 100

Jussila, Toni 201

Kandasamy, Nagaragan 369
Kleine Büning, Hans 244
Kojevnikov, Arist 70
Kröning, Daniel 201
Kullmann, Oliver 314
Kumarasubramanian, Abishek 377

Langlois, Marina 80
Larrosa, Javier 41

Levy, Jordi 10
Li, Chu Min 121
Lukasiewycz, Martin 56
Lynce, Inês 22

Makino, Kazuhisa 187
Manolios, Panagiotis 4
Manyà, Felip 10, 28
Marques-Silva, Joao 22
Middeldorp, Aart 340
Mishchenko, Alan 272
Mitchell, David 300

Nadel, Alexander 287
Navarro-Pérez, Juan Antonio 3

Oliveras, Albert 41

Pipatsrisawat, Knot 294
Porschen, Stefan 173
Prestwich, Steven 107

Sabharwal, Ashish 100
Saïs, Lakhdar 16
Samer, Marko 230
Samulowitz, Horst 215
Scheder, Dominik 148
Schneider-Kamp, Peter 340
Sebastiani, Roberto 334
Selman, Bart 100
Sinz, Carsten 201
Sloan, Robert H. 80
Sörensson, Niklas 272
Speckenmeyer, Ewald 173
Staber, Stefan 355
Szeider, Stefan 94, 230

Tamaki, Suguru 187
Teich, Jürgen 56
Thiemann, René 340
Turán, György 80

Urquhart, Alasdair 159

Van Gelder, Allen 328
van Maaren, Hans 134, 258
Venkatesan, Ramarathnam 377

Voronkov, Andrei 3
Vroon, Daron 4

Wei, Wanxia 121
Wintersteiger, Christoph M. 201

Yamamoto, Masaki 187

Zankl, Harald 340
Zhang, Harry 121
Zumstein, Philipp 148