TKK Dissertations in Information and Computer Science
Espoo 2008 TKK-ICS-D10

STRUCTURE-BASED SATISFIABILITY CHECKING

Analyzing and Harnessing the Potential

Matti Järvisalo

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Faculty of Information and Natural Sciences, for public examination and debate in Auditorium T2 at Helsinki University of Technology (Espoo, Finland) on the 29th of November, 2008, at 10 o'clock.

Helsinki University of Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science

Teknillinen korkeakoulu
Informaatio- ja luonnontieteiden tiedekunta
Tietojenkäsittelytieteen laitos

Distribution:
Helsinki University of Technology
Faculty of Information and Natural Sciences
Department of Information and Computer Science
P.O.Box 5400
FI-02015 TKK
FINLAND
URL: http://ics.tkk.fi
Tel. +358 9 451 1
Fax +358 9 451 3369
E-mail: series@ics.tkk.fi

ISBN 978-951-22-9641-5 (Print)
ISBN 978-951-22-9642-2 (Online)
ISSN 1797-5050 (Print)
ISSN 1797-5069 (Online)
URL: http://lib.tkk.fi/Diss/2008/isbn9789512296422/

Multiprint Oy
Espoo 2008

ABSTRACT: Constraint satisfaction deals with developing automated techniques for solving computationally hard problems in a declarative fashion. The main emphasis of this thesis is on constraint satisfaction techniques for the propositional satisfiability problem (SAT).

As solving techniques for propositional satisfiability have rapidly progressed during the last 15 years, implementations of decision procedures for SAT, so called SAT solvers, have been found to be extremely efficient as back-end search engines in solving large industrial-scale combinatorial problems. Since SAT solvers have become a standard tool for solving various real-world problem instances of increasing size and difficulty, there is a demand for more and more robust and efficient solvers. For understanding the success (and failures) of SAT solving in specific problem domains, it is important to investigate how different types of structural properties of SAT instances are related to the efficiency of solving the instances with different SAT-based constraint satisfaction techniques. This is the underlying motivation for this thesis.

The emphasis of the thesis is on search-based SAT solving techniques for solving structured real-world problems. The work focuses on selected topics related to the analysis and development of both complete and stochastic local search methods for SAT. Both experimental and proof complexity theoretic approaches are applied. The main contributions are three-fold.

The work contributes to the analysis of structure-based branching heuristics. A proof complexity theoretic power hierarchy is established for DPLL and clause learning SAT solvers with various structure-based branching restrictions. The proof complexity theoretic results are complemented by a detailed experimental evaluation of the effects of structure-based branching on state-of-the-art SAT solvers. In connection with structure-based branching in SAT, the work introduces the Extended ASP Tableaux proof system in the context of answer set programming, which is a field closely related to SAT solving.

The work also contributes to the development of stochastic local search methods for structured real-world SAT instances. A novel non-clausal local search method for SAT is developed by incorporating the concept of justification frontiers previously applied in the context of complete non-clausal solvers. Variants of the method are analyzed with respect approximate completeness, and adaptive noise mechanisms aimed specifically for the method are developed.

As a third point of view to structure in SAT, the work addresses the problem of generating hard satisfiable SAT instances for both DPLL-based and local search solvers by introducing the regular XORSAT model. Additionally, techniques for applying XORSAT instances specifically for benchmarking equivalence reasoning techniques in SAT solvers are developed.

KEYWORDS: propositional satisfiability, SAT, clause learning, DPLL, stochastic local search, branching heuristics, proof complexity, Boolean circuits, non-clausal formulas, problem structure, backdoors, hard instances, adaptive noise strategies, probabilistically approximately complete, answer set programming

TIIVISTELMÄ: Rajoiteratkaisimet ovat automatisoituja työkaluja, jotka mahdollistavat laskennallisesti vaikeiden ongelmien ratkaisemisen deklaratiivisesti. Tämä työ käsittelee erityisesti lauselogiikan toteutuvuusongelman (SAT) rajoiteratkaisinmenetelmiä, SAT-ratkaisimia.

Lauselogiikan toteutuvuusongelman ratkaisumenetelmien kehityksessä on saavutettu huomattavia edistysaskeleita viimeisten 15 vuoden aikana. Nykyään SAT-ratkaisimet tarjoavat erittäin kilpailukykyisen tavan ratkaista laajoja teollisuuslähtöisiä kombinatorisia ongelmia. Ongelmainstanssien kasvaessa ja sitä myötä vaikeutuessa ratkaisumenetelmien tehokkuudelle asetetaan jatkuvasti kovenevia vaatimuksia. Teollisuuslähtöisten ongelmien rakenteellisten ominaisuuksien ja näiden ongelmien ratkaisemisen haastavuuden välisen suhteen ymmärtäminen on keskeisessä osassa tehokkaiden ratkaisumenetelmien kehittämistyössä. Ongelmien rakenteen ja ratkaisutekniikoiden välisen suhteen syvällinen ymmärtäminen ja tämän tietämyksen hyödyntäminen on haastava tutkimusongelma, johon myös tämä työ keskittyy.

Työssä keskitytään erityisesti rakenteellisille teollisuuslähtöisille ongelmille suunnattujen hakupohjaisten SAT-ratkaisimien analysointiin ja kehittämiseen. Työssä analysoidaan täydellisiä SAT-ratkaisumenetelmiä sekä kokeellisesti että matemaattisesti. Lisäksi kehitetään ongelmarakenteen huomioonottavia satunnaistettuja paikallishakumenetelmiä. Työn tulokset jakautuvat kolmeen osaan.

Työssä analysoidaan ongelmien rakenteeseen pohjautuvia, täydellisissä hakumenetelmissä käytettäviä heuristiikkoja. Saavutetut todistuskompleksisuusteoreettiset tulokset koskevat tyypillisien SAT-ratkaisimille ehdotettujen rakennepohjaisten heuristiikkojen suhteellista tehokkuutta. Teoreettisten tulosten lisäksi esitetään kattava kokeellinen tarkastelu rakennepohjaisten hakuheuristiikkojen vaikutuksesta klausuulioppivien SAT-ratkaisimien toiminnallisuuteen. SAT-ratkaisimien hakuheuristiikkojen analysoinnin lisäksi työssä esitetään redundanttia ongelmarakennetta koskeva tarkastelu SAT-ratkaisimiin läheisesti liittyvällä rajoiteohjelmointialuella, jota kutsutaan vastausjoukko-ohjelmoinniksi.

Työssä kehitetään myös uudentyylinen satunnaistettu paikallishakumenetelmä. Menetelmä perustuu työssä esiteltävään ongelmien rakenteeseen perustuvaan hakuheuristiikkaan. Menetelmän kokeellisen arvioinnin lisäksi analysoidaan menetelmän eri variaatioiden likimääräistä täydellisyyttä, ja kehitetään erityisesti tälle menetelmälle soveltuvia satunnaisuuden mukautusstrategioita, jotka säätelevät haussa käytettävän satunnaisuuden määrää ajonaikaisesti.

Lisäksi työssä kehitetään SAT-ongelmainstansseja, jotka ovat rakenteellisiin ominaisuuksiinsa perustuen erittäin haastavia tyypillisille hakupohjaisille SAT-ratkaisimille. Kehitettävät instanssit soveltuvat sekä täydellisten että paikallishakutekniikoihin perustuvien SAT-ratkaisimien kokeelliseen arviointiin.

AVAINSANAT: lauselogiikan toteutuvuusongelma, SAT, klausuulioppiminen, DPLL, satunnaistettu paikallishaku, hakuheuristiikat, todistuskompleksisuus, Boolen piirit, lauselogiikan lauseiden yleinen muoto, ongelmien rakenne, vaikeat ongelmainstanssit, satunnaisuuden mukautusstrategiat, likimääräinen täydellisyys, vastausjoukko-ohjelmointi

Contents

Preface **ix**

Overview of the Thesis **xi**
Brief Summary of the Articles xii
Contributions of the Author xiii

1 Introduction **1**
1.1 Topics of this Thesis 3
Topic 1: Analysis of Structure-Based Branching in DPLL-Style Solvers 6
Topic 2: Development of SLS Methods for Structured SAT Instances 8
Topic 3: Generating Empirically Hard Satisfiable CNF Instances 9
1.2 Summary of Contributions 10

2 Propositional Satisfiability and Normal Logic Programs **14**
2.1 Constrained Boolean Circuits 14
Note on Representation Forms for Structured Formulas . . . 16
2.2 CNF SAT 17
2.3 Translating Boolean Circuits to CNF 18
2.4 Normal Logic Programs and Stable Models 18
2.5 Relationship between SAT and ASP 19

3 Proof Systems and Solver Techniques for SAT **21**
3.1 Propositional Proof Systems and Complexity 21
3.2 Resolution 22
3.3 The Davis–Putnam–Logemann–Loveland Procedure 22
Implication Graphs 23
3.4 DPLL with Clause Learning and Modern SAT Solvers 24
Conflict Graphs and Conflict Analysis 25
Implication Points, Conflict-Driven Backjumping, and CL . . 26
Restarts and the CL– Proof System 27
3.5 Circuit-Level DPLL and CL 28
3.6 Stochastic Local Search Procedures for SAT 29

4 Overview of Main Results **31**
4.1 P1: Hard Structured Satisfiable CNF SAT Instances 32
The Regular XORSAT Construction 32
Experiments 34
Motivating the Regular XORSAT Model 35
Schemes for Introducing Nonlinearity 37
Further Developments 38
4.2 P2: Structure-Based Branching in Practice 39
Effects of Input-Restricted Branching 39
Relaxed Structure-Based Branching Restrictions 42
Summary 45

4.3 P3 and P4: Proof Complexity Theoretic Limitations of Structure-Based Branching . . . 47
Input-Restricted Branching CL . . . 47
Top-Down Branching Variants of DPLL and CL . . . 49
4.4 P5: Extended ASP Tableaux and Redundancy in ASP . . . 53
Extended ASP Tableaux and Extended Resolution . . . 55
The Extension Rule and Well-Founded Deduction . . . 55
Experiments . . . 55
4.5 P6 and P7: Structure-Based Techniques for Stochastic Local Search . . . 58
BC SLS . . . 58
Comparison with CNF-Level SLS Methods . . . 60
Comparison with Non-Clausal Methods . . . 61
On Probabilistically Approximate Completeness . . . 62
Experimental Results Related to P6 . . . 63
Experiments with Adaptive Noise Strategies for BC SLS . . . 64
Dynamic Waiting Periods . . . 67

5 Conclusions **68**
5.1 Topics for Further Work . . . 69

References **72**

PREFACE

This thesis results from research I have conducted during 2004-2008 in the Computational Logic group of Laboratory for Theoretical Computer Science (Department of Information and Computer Science as of 2008) at Helsinki University of Technology (TKK). The official opponent, pre-examiners, supervisor, and instructor of this work are:

Opponent	Professor Bart Selman Cornell University, USA
Pre-examiners	Professor Paul Beame University of Washington, USA
	Professor João Marques-Silva University of Southampton, UK
Supervisor	Professor Ilkka Niemelä, TKK
Instructor	D.Sc.(Tech.) Tommi Junttila, TKK

I am honored to have Professor Selman, Professor Beame and Professor Marques-Silva as the official examiners of the thesis. I thank them all for taking the time to read this work, Professor Selman for the additional trouble of acting as the opponent at TKK, and Professor Beame and Professor Marques-Silva for the constructive pre-examination reviews.

I have been fortunate to have Professor Ilkka Niemelä as the supervisor and Dr. Tommi Junttila as the instructor of my doctoral work. Our fruitful collaboration has been a source of great pleasure. In addition to Ilkka and Tommi, I have had the pleasure of working with (in alphabetical order) Dr. Harri Haanpää, Dr. Petteri Kaski, and Dr. Emilia Oikarinen on topics related to this thesis. I would like to thank Emilia also for taking time to give comments on drafts of this thesis summary.

The main sources of funding for this work have been Helsinki Graduate School in Computer Science and Engineering HeCSE (through a funded graduate school position during 2005–2008) and the former Department of Computer Science and Engineering of TKK (through funding for postgraduate studies during 2004–2005). Additional funding was provided by Academy of Finland under Professor Niemelä's projects *Methods for Constructing and Solving Large Constraint Models* (#122399) and *Advanced Constraint Programming Techniques for Large Structured Problems* (#211025). The work has also been supported by Emil Aaltonen Foundation, Jenny and Antti Wihuri Foundation, Nokia Foundation, and Finnish Foundation for Technology Promotion TES through personal grants for doctoral studies. For traveling, HeCSE generously provided support for attending several conferences (such as CP'06, RCRA'07, AAAI'08, ECAI'08) and summer schools

(SFM:HV'06, ACAI'07, ESSLLI'08) during my studies. Additionally, Association for Constraint Programming (ACP) and American Association for Artificial Intelligence (AAAI) provided support for attending the CP'07 and AAAI'06 conferences, respectively.

Last but not even close to least, I want to thank my family, loved-ones, and friends, for providing all the non-scientific support needed, each in their individual ways.

November 2008,

Matti Järvisalo

OVERVIEW OF THE THESIS

This thesis consists of the present summary and the following articles **P1–P7**.

P1 Hard Satisfiable Clause Sets for Benchmarking Equivalence Reasoning Techniques.
Harri Haanpää, Matti Järvisalo, Petteri Kaski, and Ilkka Niemelä.
Journal on Satisfiability, Boolean Modeling and Computation, 2(1–4):27–46, 2006.

P2 The Effect of Structural Branching on the Efficiency of Clause Learning SAT Solving: An Experimental Study.
Matti Järvisalo and Ilkka Niemelä.
Journal of Algorithms: Algorithms in Cognition, Informatics, and Logic, 63(1–3):90–113, 2008.

P3 Limitations of Restricted Branching in Clause Learning.
Matti Järvisalo and Tommi Junttila.
Constraints, to appear (2008). 29 pages.

P4 On the Power of Top-Down Branching Heuristics.
Matti Järvisalo and Tommi Junttila.
In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the 23rd AAAI Conference on Artificial Intelligence* (AAAI 2008), pages 304–309. AAAI Press, 2008.

P5 Extended ASP Tableaux and Rule Redundancy in Normal Logic Programs.
Matti Järvisalo and Emilia Oikarinen.
Theory and Practice of Logic Programming, to appear (2008). 27 pages.

P6 Justification-Based Non-Clausal Local Search for SAT.
Matti Järvisalo, Tommi Junttila, and Ilkka Niemelä.
In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fanotakis, and Nikos Avoukis, editors, *Proceedings of the 18th European Conference on Artificial Intelligence* (ECAI 2008), volume 178 of *Frontier in Artificial Intelligence and Applications*, pages 535–539. IOS Press, 2008.

P7 Justification-Based Local Search with Adaptive Noise Strategies.
Matti Järvisalo, Tommi Junttila, and Ilkka Niemelä.
In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence and Reasoning* (LPAR 2008), volume 5330 of Lecture Notes in Computer Science. Springer, 2008.

P1: A novel family of empirically hard structured satisfiable SAT instances is introduced, and schemes for introducing nonlinearity to the instances are developed. This makes the instances suitable for benchmarking solvers with equivalence reasoning techniques. An extensive experimental evaluation shows that state-of-the-art solvers scale exponentially in the instance size. Compared to other well-known families of satisfiable benchmark instances, the presented instance family is among the hardest.

P2: An extensive experimental evaluation of the effects of structure-based branching restrictions on the efficiency of solving structured SAT instances is presented. The work provides a thorough analysis of the effect of branching restrictions on the inner workings of a state-of-the-art clause learning SAT solver, going deeper than merely measuring the solution time. Additionally, relaxed branching restrictions based on structural properties of SAT instances are considered. A preliminary version [123] of this article was presented at the 14th RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA'07).

P3: The proof complexity theoretic effect of statically restricting branching to so called input variables is studied for the widely applied DPLL method. The main result is that, even with unlimited restarts, input-restricted branching clause learning DPLL cannot simulate DPLL. A preliminary version [124] of this article was presented at the 13th International Conference on Principles and Practice of Constraint Programming (CP'07).

P4: The relative best-case performance of DPLL-based structure-aware SAT solvers is studied in terms of the power of the underlying proof systems. The systems result from (i) varying the style of branching and (ii) enforcing dynamic restrictions on the decision heuristics. Considering DPLL both with and without clause learning, a proof complexity theoretic efficiency hierarchy is presented for refinements of DPLL resulting from combinations of decision heuristics and branching styles.

P5: In analogy with the Extended Resolution proof system for SAT, an extended tableau calculus for answer set programming (ASP) is introduced. The efficiency of Extended ASP Tableaux is related to Extended Resolution. Closely related to Extended ASP Tableaux, the effect of redundant rules on the efficiency of ASP solving is experimentally investigated. A preliminary version [126] of this article was presented at the 23rd International Conference on Logic Programming (ICLP'07) and received the ICLP'07 Best Student Paper Award.

P6: Novel stochastic local search (SLS) techniques for solving structured SAT instances are developed. By harnessing the concept of justification frontiers, new SLS heuristics are introduced which concentrate

the search into relevant parts of instances, exploit observability don't cares and allow for an early stopping criterion.

P7: The BC SLS approach developed in **P6** is extended in two ways. Probabilistically approximately complete (PAC) variants of BC SLS are devised. Additionally, adaptive noise mechanisms for BC SLS are developed, including mechanisms based on dynamically adapting the waiting period for noise increases. This article was also presented at the 2nd International Workshop on Logic and Search (LaSh'08).

Contributions of the Author

P1: The main idea behind the regular XORSAT construction is due to Kaski, who also played a major role in the analysis presented in Section 3.2 of the article. The schemes presented in Section 3.3 of the article are mainly joint work of the author with Niemelä. The experiments were designed jointly with the co-authors. The experimental work was implemented, carried out, and reported by the author.

P2: The hypotheses and experiments were designed jointly with the co-author. The experimental work was implemented, carried out, analyzed, and reported by the author.

P3: The theoretical part of the article is joint work with the co-author. The initial idea to apply known results on extended resolution in the proof constructions is by the author. The experimental work, which is extended further in **P2**, was implemented, carried out, and reported by the author.

P4: As a continuation of **P3**, the article is joint work with the co-author. The idea to relate known results on analytic tableaux is by the author. The modification to a construction from **P3**, as illustrated in Figure 4 in **P4**, is by the co-author.

P5: The work was initiated by the author, the ideas for the content being mainly due to the author. The theoretical part of the article is joint work with the co-author. The experimental work was designed, carried out, and reported by the author.

P6: The main algorithm is joint work with the co-authors. The experimental work was implemented, carried out, and reported by the author.

P7: The main algorithm, which is a modification of the algorithm presented in **P6**, aimed at achieving approximate completeness, is joint work with the co-authors. The new variants of adaptive noise are due to the author. The author designed, carried out, and reported the experiments, with the exception that the running times reported in the work are from an implementation of the method by Junttila.

1 INTRODUCTION

The field of constraint satisfaction lies in the intersection of computer science, artificial intelligence, and mathematics. It deals with developing automated techniques for solving computationally hard problems. The problem domains where constraint satisfaction techniques are applied are typically characterized by computational intractability: the underlying decision problems are often **NP**-complete [80] or even harder.

Problem solving through constraint satisfaction is of declarative nature, consisting of two parts: *modelling* and *solving*. The task of modelling is to describe the problem to be solved in a chosen mathematical formalism (a *constraint* or *modelling language*) by describing relationships between variables of the problem through posing constraints over the possible value assignments for the variables. The result of modelling is a general method referred to as the *translation* of the domain-specific problem in the modelling language. The output of a translation for a particular input (problem instance of the original problem) is referred to as the *encoding* (of the particular input). An important property of a translation is that a solution to an arbitrary instance I of the original problem can be easily extracted from any value assignment for the variables respecting the constraints in the encoding of I. The goal of modelling is to provide such an translation. The task of *solving*, on the other hand, is to find such a value assignment (a *solution* to the encoding of the instance), if one exists.

First-order logic provides characteristic modelling formalisms for problems of various complexity. Since the decision problem of propositional satisfiability (SAT)—given a propositional logic formula, we are asked whether there is a satisfying truth assignment for the formula—is **NP**-complete [58], the language of propositional logic, a subset of first-order logic, provides a choice for expressing problems in **NP**.

The main emphasis of this thesis is on constraint satisfaction techniques for SAT, or SAT *solving*. As propositional logic provides a succinct representation for studying computational complexity, SAT being an archetypical **NP**-complete problem, interest in SAT is historically mostly of theoretical nature. However, as the solving techniques for propositional satisfiability have rapidly progressed during the last 15 years, implementations of decision procedures for SAT (SAT *solvers*) [98] have been found to be extremely efficient as back-end solving engines for industrial-scale combinatorial problems. Typical examples of such real-world application domains of SAT solvers include automated planning [139, 140, 195, 136, 54], bounded model checking (BMC) of hardware and software [38, 39, 148, 20, 171], and electronic design automation applications such as automated test pattern generation (ATPG) [151, 224, 228] and symbolic trajectory evaluation (STE) [198]. Additional recent application areas of SAT solving techniques include problems related to diagnosis and diagnosability testing of discrete-event systems [194, 103], bioinformatics [170, 231], logical cryptanalysis [175, 178, 66, 73], and model checking of security protocols [19].

The simple formalism of propositional logic has greatly contributed to the current successes of SAT-based problem solving. On one hand, the simplicity of the formalism allows for highly efficient solvers to be imple-

mented through efficient data structures and lean design. On the other hand, in addition to the fact that problems in **NP** allow for efficient translations (reductions) into SAT, very succinct translations (resulting even in linear size encodings with small coefficients) are known for various problem domains, which is highly relevant for practical purposes. An additional practically relevant advantage of typical SAT solvers aimed at solving real-world problems today is their ability—in addition to deciding whether an satisfying truth assignment for a given SAT instance exists—to construct such a solution. Furthermore, while SAT does not capture **PSPACE** [186], SAT-based approaches have been very successful for the already mentioned **PSPACE**-complete problems of automated planning and model checking. This is achieved by restricting the length of solutions sought to polynomial ones in order to obtain an **NP**-complete problem. For example, in SAT-based planning [139] one restricts to polynomial plans; for model checking, this leads similarly to BMC, *bounded* model checking [38].

When solving problems with SAT solvers, the propositional formula encoding the original problem instance is typically translated into *conjunctive normal norm* (CNF, or *clausal* form) [222], that is, into a conjunction of disjunctions of Boolean variables and their negations. In this sense, SAT can be seen as a special case of finite-domain *constraint satisfaction problems* (CSPs) [199, 68]. Generally speaking, the language of CSPs allows one to express conjunctions of arbitrary constraints over variables with given domains, where individual constraints are seen as relations that express the feasible value combinations for the specific constraint. Hence, CNF SAT is the special case of CSPs in which only conjunctions of clausal constraints on variables with binary domains are allowed. This results in the fact that, in many cases, advances in understanding and developing solving techniques for SAT can have implications to the study of constraint satisfaction more generally. The connection of general CSPs [199, 68, 44] and SAT is further highlighted by studies on exploiting SAT techniques for solving CSPs, as exemplified for instance by [191, 23, 241].

Successfully applied techniques for solving SAT can be divided into two ideologically different approaches: those based on *knowledge compilation* (such as [215, 51, 63]) and the *search-based* approaches. In this work the focus is on search-based SAT solving techniques. In contrast to compilation-based approaches, search-based procedures developed for SAT concentrate on determining satisfiability alone, providing a satisfying truth assignment if one exists. By concentrating on satisfiability alone, the problems of exponential space consumption often experienced in knowledge complication based approaches can be avoided, which is a key factor in the success of search-based SAT solvers.

The most successful SAT solvers aimed at solving structured problems today are based on the complete *Davis–Putnam–Logemann–Loveland* procedure (DPLL) [65, 64]. Such solvers perform an intelligent search over the whole assignment space (or *search space*) through backtracking. A highly relevant aspect of state-of-the-art implementation of DPLL-based SAT solvers is that they typically are "black boxes", or in other words, such solvers require no handpicked parameters from the user.

The relevance of DPLL solvers is nowadays further highlighted by their

application as a core solver engine for solving the **#P**-complete [237] problem of *model counting* [129, 206, 99, 96, 147] which has applications in the field of probabilistic reasoning [24, 165, 207], as well as in the *Satisfiability Modulo Theories* (SMT) [209, 183, 46] approach, in which the modelling language is enriched with more expressive constraint types, such as *linear (in)equations*, to allow Boolean combinations of such constraints. A further motivating example of the importance of SAT solver techniques is their close relation to solver techniques applied in the rule-based constraint programming paradigm of *answer set programming* (ASP) [182, 87, 26], which is a form of non-monotonic reasoning [50] under the *stable model semantics* [88, 86]. Due to the close relationship between stable models and satisfying truth assignments in propositional logic, many translations of ASP as SAT have been developed [35, 121], some of which are suited for solving ASP as SAT using an incremental approach based on SAT solvers [164, 162]. Most interestingly, however, due to the close relationship between ASP and SAT, the typical ASP solvers [220, 17, 153, 82] share several features with successful DPLL-based SAT solvers. This in turn implies that further advances in SAT solvers can contribute to more efficient solvers for ASP, as well. Yet another example of the relevance of DPLL-based solvers are the various solvers developed for solving instances of the **PSPACE**-complete problem of QBF satisfiability by extending DPLL-based techniques [53, 155, 245, 205, 93].

In contrast to complete search procedures, stochastic local search (SLS) methods for SAT (see [218, 213, 217, 177, 115, 114, 212, 111] for examples, and [116] for a general view) are typically based on iterating over solution candidates by *flipping* value assignments in the current candidate based on a *neighborhood* function. Such procedures are characterized by their inability to show (generate a *proof* for) the non-existence of solutions, although recently also local search for *unsatisfiability* has been considered [189]. While SLS SAT solvers have proven very successful in solving *random* satisfiability problem instances (see [211, 49] for examples), the breakthroughs in applying SAT solvers in real-world problem domains are due to DPLL-style complete SAT solvers.

1.1 Topics of this Thesis

This work focuses on selected topics related to the experimental and theoretical analysis and development of both complete and stochastic local search methods for SAT. We will now introduce the main topics addressed in this thesis.

As search-based SAT solvers have become a standard tool for solving various real-world problem instances of increasing size and difficulty, there is demand for more and more robust and efficient solvers. For understanding the success (and failures) of SAT solving in structured problem domains, it is important to investigate how different types of structural properties of SAT instances are related to the efficiency of solving instances using SAT-based constraint satisfaction techniques. This is the underlying motivation for this thesis. The emphasis of the thesis is on search-based SAT solving techniques for solving structured real-world problems.

Knowledge of different types of structural properties of SAT instances is

related to the efficiency of solving such instances. For example, the experimentally noticed inefficiency of DPLL on CNF SAT instances based on systems of linear equations modulo two, referred to as XORSAT, has motivated work on developing additional techniques for DPLL in attempt to lift the efficiency of DPLL on such instances [28, 109, 157, 242]. The contribution on CNF-level SAT in this thesis addresses the problem of generating benchmark instances which are extremely difficult to solve with both DPLL-based and SLS search techniques.

On the other hand, the study of structure in SAT is closely related to structural representation forms of SAT instances. Typically SAT solvers—both DPLL-based and local search procedures—restrict their input to CNF. However, in addition to difficulty of modelling problems directly in CNF, a major problem in using CNF encodings is that structural properties, such as functional dependencies, of the original problem domain are not directly evident from the resulting CNF formula. Compared to CNF, more natural representations for arbitrary propositional formulas are often used during modelling. The problem at hand is typically encoded as a general propositional formula ϕ, which is then translated into an equi-satisfiable CNF formula. A "standard" linear time translation [233] is typically applied, which achieves a linear size CNF encoding of any propositional formula by introducing additional variables for representing the subformula structure of the original formula. *Boolean circuits*—and refined notions [149, 1]—provide a natural, structure-preserving representation form (sometimes referred to as *non-clausal formulas*) for modelling typical structured SAT problems as general propositional formulas.

There has been work on recovering circuit-level representation from CNF problem encodings [104, 200, 77]. Unfortunately, this is not a trivial task. However, by explicitly maintaining the Boolean circuit representation of the problem encoding, structural properties of general propositional formulas can be easily detected directly from the structure-preserving representation. There is a wide body of work on lifting the DPLL procedure to work directly on Boolean circuits, see [132, 149, 79, 119, 230] for instance. This enables the development of new solver techniques that attempt to exploit the structural knowledge. Additionally, a few SLS methods have also been proposed for general propositional (non-clausal) formulas [208, 137, 187, 223].

Motivated by these consideration, a central part of this thesis concentrates on analyzing and developing search-based solver techniques for general propositional formulas. In more detail, the topics addressed in this thesis are the following.

Topic 1: Analysis of Structure-Based Branching in DPLL-Style Solvers

Branching heuristics, that is, deciding on which variable to next set a value during search, play an important role in the efficiency of search. Techniques for making effective decisions during search are vital. One way for circuit-level solvers to exploit the explicit structured representation is to use it for guiding branching heuristics. Intuitively, the inherent structure of the problem domain is reflected in individual variables in the SAT encoding, and making decisions on structurally irrelevant variables may have an exponential effect on the running times of SAT

solvers.

From the theoretical point of view, in this work we investigate the *best-case* performance of SAT solving procedures using different structure-based branching heuristics through *proof complexity* [60, 235, 34]. Proof complexity gives means of studying the relative power of the inference systems (or *proof systems*) underlying SAT solvers in terms of the shortest possible proofs in the systems.

The theoretical results presented in this thesis reveal inherent weaknesses in various structure-based branching heuristics suggested in the literature for DPLL-based SAT solvers. Additionally, the theoretical results are complemented by a detailed experimental evaluation of the effects of structure-based branching on state-of-the-art SAT solvers.

In connection with structure-based branching in SAT, this work introduces the Extended ASP Tableaux proof system in the context of answer set programming, which is a field closely related to SAT solving. This part of the thesis brings ideas from the powerful *extended resolution* proof system for SAT [233] to the context of ASP.

Topic 2: Development of SLS Methods for Structured SAT Instances

Stochastic local search techniques for SAT are today not competitive with DPLL on real-world problems. In fact, in the late 90's there was a shift from SLS solvers to DPLL-based solvers as the dominating approach to search-based SAT solving. For widening the applicability of SLS methods, further work on improving SLS techniques for structural problems is needed. In particular, developing techniques for handling variable dependencies efficiently has been identified as a major challenge [141].

This thesis contributes to the development of stochastic local search methods for structured real-world SAT instances. A novel non-clausal local search method for SAT is developed. The method works directly on the level of Boolean circuits, and applies techniques for handling variable dependencies in a new way in the context of SLS.

Topic 3: Generating Empirically Hard Satisfiable SAT Instances

For many applications areas, the SAT instances of interest are satisfiable, and the key task of a SAT solver is to find a satisfying truth assignment [40, 57, 140, 151]. Hence, good performance for satisfiable instances is very important in practice. In fact, there has recently been a lot of interest in generating *satisfiable* problem instances that are *empirically* hard for both complete and SLS SAT solvers. Structurally interesting empirically hard satisfiable instance families are especially useful in developing effective heuristics for satisfiable instances, and, in particular, only satisfiable instances are relevant for benchmarking incomplete SLS methods.

As a contribution to Topic 3, this thesis develops techniques for generating empirically hard satisfiable benchmarks. Additionally, techniques

for applying XORSAT-style CNF SAT instances specifically for benchmarking equivalence reasoning techniques in SAT solvers are developed.

We will next give general overviews of these three topics with related work. A concise summary of the contributions to each to the topics is then provided in Section 1.2.

Topic 1: Analysis of Structure-Based Branching in DPLL-Style Solvers
There has been a significant amount of work on boosting the efficiency of DPLL solvers. *Clause learning* [174, 33] can be regarded as the most important progressive step in the effectiveness of SAT solvers in structured problem domains, as witnessed by a sequence of further improved solvers [130, 174, 180, 97, 72], and also by theoretical analysis [33].

Perhaps one of the most studied proof systems for SAT from the perspective of proof complexity are (*unrestricted* or *general*) resolution [197] and its *refinements* (see [52], for instance). Interestingly, there is a tight connection between resolution and DPLL: it is well-known that DPLL is equivalent to a refinement of resolution called *tree-like resolution*. Recently clause learning DPLL has also been characterized as a proof system called CL [33]. Through this characterization, Beame et al. [33] show that CL can provide exponentially shorter proofs than DPLL. In other words, DPLL cannot *polynomially simulate* [60] CL. This result gives the first theoretical explanation for the practical efficiency of implemented SAT solvers incorporating clause learning.

This thesis makes several contributions relating problem structure and branching heuristics in DPLL-style solvers. A major part of the results deals with experimental and proof complexity theoretical analysis of the effect of applying structure-based branching heuristics on the efficiency of DPLL and CL on the level of Boolean circuits.

Structure-Based Branching in DPLL and Clause Learning

A major part of this work considers structure-based *branching restrictions*, in which branching in a SAT solver is restricted—either *statically* or *dynamically*—to a subset of variables. The considered branching restrictions are based on structural properties which are explicit in the Boolean circuit representation of the instance at hand.

The idea behind branching restrictions is to limit the set of variables the solver is allowed to branch on to a small subset I instead of the set N of all variables in the SAT instance at hand. The solver will then apply its own dynamic heuristics on the variables in I. In static branching restrictions the subset of variables I stays invariant for the whole duration of search. With dynamic branching restrictions the set I is varied during search. It should be noted that *branching variable orderings* for DPLL based on structural information have also been studied [118, 12]. In contrast to the branching restrictions studied in this thesis, in these works the solver is forced to follow an order derived from structural properties of the formula.

The motivation behind static branching restrictions is that, by selecting I so that the solver remains complete, the search space size is radically reduced from the order of $2^{|N|}$ to that of $2^{|I|}$, where $|I| \ll |N|$. In the context of

DPLL, the concept of a *(strong) backdoor set* [243, 102, 201] of variables is closely related to restricting branching: a *unit propagation backdoor* is a set of variables such that, once all of these variables have values, all the other variables are set values by unit propagation.

Although the original motivation for studying backdoors comes from bringing insights into *heavy-tailed behavior* in combinatorial search [100, 102], a practical way of exploiting backdoor sets would be to attempt to improve DPLL search efficiency by *backdoor-based branching*. In other words, one can restrict a DPLL SAT solver to branch only on variables in a unit propagation backdoor. However, deciding whether a backdoor set of a given size exists is intractable in general [227, 69]. With this in mind, knowledge of the underlying structural properties of variables in the instance at hand makes it easier to apply branching restrictions when solving the instance.

An example of a natural branching restriction is provided by the set of so called *input* (or *independent*) variables. In SAT-based approaches to structured problems such as bounded model checking and automated planning, the CNF encoding is often derived from a transition relation, where the behavior of the underlying system is dependent on the *input*—initial state, nondeterministic choices due to the external environment, et cetera—of the system. Problems such as ATPG that deal with logical circuit designs serve as additional examples of domains where system input is naturally present. By noticing that the system behavior is determined by its input, it is in fact the case that all variables in the SAT encoding of the system can be assigned through unit propagation once all input variables have been assigned values. In other words, the set of input variables is a strong unit propagation backdoor set—although possibly not of *minimum* cardinality. Hence DPLL remains complete even if branching is restricted to the set of input variables alone. In fact, experimental case studies in specific problem domains [92, 225, 91] have shown that in some cases SAT solvers benefit from restricting the variables the solver is allowed to branch on to those variables that model the input of the underlying system.

In contrast to static branching restrictions, in dynamic branching restrictions the set on which branching is restricted to is varied during search. One applied heuristic idea is to apply branching in a *top-down* fashion, starting from the constraints imposed on the output gates of the circuit, and to search for *justification* for the currently imposed values [150, 166]. This is closely related to the tableau decomposition rules applied in analytic tableaux [222]. A modification to the actual *style of branching* in DPLL-based algorithms, aiming at eagerly justifying the currently unjustified gates, has also been considered [149].

In this thesis the effects of input-restricted and top-down branching on DPLL and CL are studied from a proof complexity perspective. Complementing these theoretical results, an experimental analysis of the effect of structure-based static branching restrictions on clause learning SAT solvers is also provided.

Structural Redundancy and Complete Search for ASP

Structured instances arising from real-world problem domains are typically large. The used propositional encoding naturally has an effect on the size

of the resulting problem instances. From the solver perspective, instance size can have a drastic effect on the effectiveness of search, especially as solvers are required to solve increasingly large problems. Working in the interface of the modelling and solving subtasks, *preprocessing techniques* [47, 169, 25, 226, 71], which act on the problem instance before actually calling the *core* solver, have been actively developed. In contrast to modelling, in preprocessing the input and output language remains the same, with the additional, natural requirement of preserving the satisfiability of the input problem instance. The aim of preprocessing is to transform the problem instance into another instance that is likely to be easier to solve for the actual solver. Typically the aim of preprocessing is to decrease the size of the problem instance—the number of variables and/or the number of constraints in the instance—with the intuition thatsmaller instances are often easier to solve than equivalent larger ones. However, as noted for example in [168], the relationship between problem size and hardness is not at all straightforward. Considering SAT, an extreme example of this puzzling relationship is provided through the *extended resolution* proof system [233]. Extended resolution is a simple extension of resolution, in which, in addition to applying the original resolution rule, one can introduce in a controlled manner so called *redundant clauses* to the CNF SAT instance at hand. Notably, this extension results in a very powerful proof system which cannot be polynomially simulated by resolution [59, 235], and thus neither by the typical DPLL-based SAT solvers today [33].

In this thesis, we exploit known results on the power of extended resolution in different ways. The proof complexity theoretic results on restricted branching in DPLL make use of this knowledge in the main proof constructions. On the other hand, similar techniques are applied in introducing studying the *Extended ASP Tableaux* proof system—in analogy with extended resolution for CNF SAT [233]—in the context of ASP.

Topic 2: Development of SLS Methods for Structured SAT Instances

While the most successful SAT solvers aimed at solving structured problems are DPLL-based, the quest for alternative solving techniques is important, especially for maintaining diversity in the study of novel solver techniques. The fact that SLS procedures often perform poorly compared to DPLL-based solvers on real-world problems has been identified to be related to the fact that the heuristic techniques in typical SLS procedures do not take into account the underlying *structural aspects* of real-world problems [141]. One problem in developing efficient techniques for handling variable dependencies is that typically the most efficient SLS solvers work on the flat CNF input format. Incorporation of elements from complete solvers into CNF-level SLS [111, 161, 160] and hybrids of complete and SLS techniques [176, 196, 106, 13, 154, 75] have been studied to some extend. However, there seems to be room for novel structure-based SLS techniques exploiting variable dependencies more directly. In contrast to abundance in complete DPLL-style non-clausal algorithms [132, 149, 150, 230], only a few SLS methods have been proposed for general propositional formulas [208, 137, 187]. Common to these SLS approaches is that they attempt to explicitly exploit variable dependencies through input variables.

This thesis develops novel non-clausal SLS methods for structured real-world SAT instances. The underlying idea in the proposed method is to drive search top-down in the Boolean circuit structure rather than focusing on input variables as previously suggested [208, 137, 187]. Motivated by *justification frontier* heuristics [150] applied in complete circuit-level SAT solvers, our search technique looks for a *justification* for the Boolean circuit instead of attempting to find a satisfying truth assignment.

Topic 3: Generating Empirically Hard Satisfiable CNF Instances

From the practical perspective, the development of structured, empirically hard SAT benchmark instances enables the experimental evaluation of new search techniques such as novel search heuristics.

Among the well-known sources of empirically hard instances are the random k-SAT [179, 61] model and its restrictions such as regular random k-SAT [45]. As observed in [179], the *clauses–to–variables ratio* parameter α characterizes the average difficulty of solving the resulting instances with DPLL. For a fixed k, a sharp *easy-hard-easy* transition happens at a specific value of α [61, 142]. At the same critical threshold value of α, a rapid transition in the fraction of satisfiable instances occurs. Below the threshold a vast majority of instances are satisfiable (have a solution), while above the threshold, most instances are unsatisfiable (without a solution), with approximately 50% of all instances satisfiable at the threshold. The relative hardness of the unsatisfiable instances, located in the region above the critical threshold at which the phase transition takes place, has also been studied from the proof complexity perspective [55].

However, the satisfiability of random k-SAT instances cannot be determined efficiently beforehand, and hence generating hard satisfiable random instances at the phase transition for benchmarking purposes does not often serve a purpose. This problem has been addressed by "hiding" solutions by generating only clauses that are satisfied by truth assignments selected beforehand [5, 27, 128]. More structured satisfiable benchmark instances have been developed based on, for example, quasigroup completion [4, 138].

Although systems of linear equations modulo 2 are polynomial-time solvable by Gaussian elimination, linear equation systems presented in CNF (XORSAT) are another well-known source of empirically hard CNF SAT instances. In the area of circuit verification and logical cryptanalysis there are problems involving linear substructure (XOR equations) that are very challenging for SAT solvers [28], which gives one motivation to considering XORSAT-style benchmark instances. The random k-XORSAT model [193] even exhibits a phase transition phenomenon similar to that of random k-SAT. Furthermore, XORSAT models enable the generation of instances with predefined satisfiability. Many of the proposed *satisfiable* XORSAT families are motivated by spin glass models from statistical physics [81, 127, 181].

Within the scope of Topic 3 in this thesis the generation of empirically hard satisfiable benchmarks is considered by combining regularity and XOR-SAT to achieve notably hard satisfiable CNF instance.

1.2 Summary of Contributions

The contributions made in this thesis divide into the three introduced topics as follows.

Topic 1: Analysis of Structure-Based Branching in DPLL-Style Solvers

Publications **P2**–**P5** deal with structure-based branching for DPLL-style complete solvers.

When applying structure-based branching restrictions in DPLL-based solvers, a natural question to ask is *whether the power of the underlying inference systems of the solvers is affected by the branching restriction*. Publications **P3** and **P4** address the effect of branching restrictions on the proof complexity theoretic power of DPLL-based solvers with and without clause learning. Both static (input-restricted) and dynamic (top-down branching variants) branching restrictions are considered. These studies give answer to the question of the relative power to the proof systems underlying such branching restricted solvers. As the main results of **P3** and **P4**, a detailed proof complexity theoretic efficiency hierarchy for such variants of DPLL and CL (DPLL with clause learning) is established. These results complement previous results on selected variations of restricted branching in DPLL (without clause learning) [125].

In more detail, in **P3** we settle the relative efficiency of input-restricted branching CL. We show that even with unlimited restarts and the ability to create conflicts at will, input-restricted CL cannot even simulate the basic DPLL *without clause learning*. This is surprising, since the unrestricted version of this variant of CL can efficiently simulate general resolution [33], being thus very powerful compared to DPLL. This implies that all implementations of CL, even with optimal heuristics, have the potential of suffering a notable efficiency decrease if branching is restricted to input variables.

In **P4** we present a relative efficiency hierarchy for variations of circuit-level DPLL and CL resulting from combinations of branching heuristics and branching styles. Motivated by ideas from solver development, we study the variations (i) DPLL-style top-down restricted, (ii) DPLL-style justification restricted [150, 166], and (iii) ATPG-style justification restricted [149] branching DPLL and CL. For example, for DPLL we establish a strict hierarchy for these variants. Perhaps the most surprising result obtained in **P4** is that CL using unlimited restarts with justification restricted decisions heuristics cannot even simulate the top-down restricted variant of DPLL. Thus, although the idea of eagerly and locally justifying the values of currently unjustified constraints is an intuitively appealing one, it can lead to dramatic losses in the best-case efficiency of a structure-aware SAT solver even when the powerful search space pruning technique of clause learning is applied.

Taking a complementary perspective to that of **P3** and **P4**, **P2** reports an extensive experimental evaluation of the effect of static branching

restriction on clause learning SAT solving. Notably extending and updating previous case studies on restricted branching [92, 225, 91, 219], we analyze in detail the effect of statically restricted branching on the effectiveness of state-of-the-art clause learning and branching heuristics. Previous studies consider mainly input-restricted branching as the only structural way of restricting the decision making in SAT solvers. In **P2** we devise and apply controlled schemes for allowing branching additionally on CNF variables other than inputs based on underlying structural properties of the problems. We relate the differences in efficiency resulting from different structural properties to the effectiveness of clause learning techniques. Since statically restricted branching can be seen as an application of backdoor-based branching (using possible non-minimal backdoor sets), the work in **P2** provides insights into the applicability of backdoor sets for restricting branching heuristics in clause learning SAT solvers. The experimental results of **P2** confirm that, in general, input-restricted branching can cause a notable loss of robustness in a clause learning SAT solver. Input-restricted branching results in, for example, longer conflict clauses on the average, which in itself makes clause learning less effective and can also hinder the overall efficiency of the solver. However, by relaxing the input-restriction by allowing branching additionally on variables with particular underlying structural properties in a systematic fashion, we are able to show that branching can in fact be restricted quite heavily without making a clause learning solver notably less efficient. Moreover, the choice of the structural property on which such a relaxation is based on does make a difference.

Exploiting known results on the power of the Extended Resolution proof system for CNF SAT, in **P5** we introduce *Extended ASP Tableaux* proof system in the context of answer set programming. The extension rule of Extended ASP Tableaux allows for adding redundant structure to ASP instances, resulting in the fact that ASP solvers, which are closely related to DPLL-based SAT solvers, may branch on these added substructures during search. This work is motivated by the fact that, while there is an abundance of studies on the proof complexity of proof systems for SAT, this has not been the case for ASP. The close relation of ASP and SAT and the respective theoretical underpinnings of practical solver techniques has also received little attention up until recently, although the fields could gain much by further studies on these connections. Hence, in this thesis, publication **P5** continues in part bridging the gap between ASP and SAT. This work complements the recent tableau-style ASP proof system characterizations [84, 85] of the inference techniques applied in state-of-the-art ASP solvers and related studies on the (proof complexity) theoretical underpinning of such solvers [16, 90, 83]. In addition to theoretical observations on Extended ASP Tableaux, we experimentally study the effect of adding redundant structure to ASP instances on the efficiency of ASP solvers.

Topic 2: Development of SLS Methods for Structured SAT Instances

Publications **P6** and **P7** address the challenge of developing SLS SAT

solvers which exploit variable dependencies in structured real-world SAT instances. We develop novel local search SAT solving techniques aimed at solving structured real-world SAT instances, directly using Boolean circuits. In contrast to previously suggested non-clausal SLS methods that focus flipping on input variables [208, 137, 187, 223], our idea is to drive local search top-down in the circuit structure. By guiding the search using justification frontiers, we enable exploiting *observability don't cares* [204] in the context of local search, drive the search to relevant parts of the circuit, and offer early stopping criteria which allow to end the search when the circuit is *de facto satisfiable* even if no concrete satisfying truth assignment has been found.

In addition to introducing BC SLS, in **P7** we analyze *probabilistically approximate completeness* (PAC) [113] of variants of the method, and achieve the PAC property while keeping the search focused. Furthermore, we develop new *adaptive noise mechanisms* [114] aimed specifically for BC SLS.

The novel techniques behind BC SLS show promise: a current implementation of BC SLS can outperform typical CNF-level SLS methods such as WalkSAT and AdaptNovelty+ in running times and in the number of moves up to multiple magnitudes of difference on real-world BMC circuit instances.

Topic 3: Generating Empirically Hard Satisfiable CNF instances

Publication **P1** addresses the problem of generating hard satisfiable CNF SAT instances for both DPLL-based and local search solvers.

We develop a satisfiable CNF family—(*random*) *regular XORSAT*—by transforming *random regular graphs* into systems of linear equations modulo 2 presented in CNF. The novelty is that we combine regularity and randomness in the context of XORSAT by employing random regular graphs—motivated by their expansion properties—to force properties which limit unit propagation as much as possible. Due to the simplicity of the model, it is easy to generate large numbers of instances of the same size. Both DPLL–based and local search state-of-the-art SAT solvers scale exponentially on regular XORSAT.

Furthermore, we develop techniques for introducing nonlinearity (other Boolean connectives than XOR) into the equation systems to make the benchmarks challenging also for CNF–level solvers equipped with equivalence reasoning techniques. By introducing nonlinearity, even DPLL solvers with equivalence reasoning techniques scale exponentially on regular XORSAT. However, we observe significant differences in the effectiveness of different equivalence reasoning techniques.

Compared to several other families of satisfiable instances, regular XORSAT is among the hardest. As suggested by the results of the SAT Competition 2005 (see `http://www.satcompetition.org/`), already small instances of regular XORSAT are very hard to solve.

While the reader is kindly requested to refer to the original publications **P1**–**P7** for particulars, a more detailed discussion of the main results is pro-

vided in Chapter 4. Before this, we will go through some needed technical background on propositional satisfiability (Chapter 2) and the main ideas behind the solving methods considered in this work (Chapter 3).

2 PROPOSITIONAL SATISFIABILITY AND NORMAL LOGIC PROGRAMS

This section reviews relevant background for discussing the main results of the thesis. Sections 2.1–2.5 provide definitions for concepts related to propositional satisfiability, constrained Boolean circuits, and normal logic programs. Additionally, the standard translations between CNF SAT and both Boolean circuits and normal logic programs under the stable model semantics are reviewed.

2.1 Constrained Boolean Circuits

A Boolean circuit over a finite set G of *gates* is a set $\mathcal{C}$ of equations of the form $g := f(g_1, \ldots, g_n)$, where $g, g_1, \ldots, g_n \in G$ and $f : \{\mathbf{f}, \mathbf{t}\}^n \to \{\mathbf{f}, \mathbf{t}\}$ is a Boolean function, with the additional requirements that (i) each $g \in G$ appears at most once as the left hand side in the equations in $\mathcal{C}$, and (ii) the underlying directed graph

$$\langle G, E(\mathcal{C}) = \{\langle g', g\rangle \in G \times G \mid g := f(\ldots, g', \ldots) \in \mathcal{C}\}\rangle$$

is acyclic. If $\langle g', g\rangle \in E(\mathcal{C})$, then g' is a *child* of g and g is a *parent* of g'. Similarly, if there is a non-empty path from a gate g' to a gate g in $\langle G, E(\mathcal{C})\rangle$, then g' is a *descendant* of g. If $g := f(g_1, \ldots, g_n)$ is in $\mathcal{C}$, then g is an f-gate (or of *type* f), otherwise it is an *input gate*. A gate with no parents is an *output gate*. A (partial) assignment for $\mathcal{C}$ is a (partial) function $\tau : G \to \{\mathbf{f}, \mathbf{t}\}$. An assignment τ is consistent with $\mathcal{C}$ if $\tau(g) = f(\tau(g_1), \ldots, \tau(g_n))$ for each $g := f(g_1, \ldots, g_n)$ in $\mathcal{C}$.

A *constrained Boolean circuit* $\mathcal{C}^\alpha$ consists of a Boolean circuit $\mathcal{C}$ and a partial assignment α for $\mathcal{C}$. With respect to a constrained circuit $\mathcal{C}^\alpha$, each $\langle g, v\rangle \in \alpha$ is a *constraint*, and g is *constrained* to v if $\langle g, v\rangle \in \alpha$. An assignment τ *satisfies* $\mathcal{C}^\alpha$ if (i) it is consistent with $\mathcal{C}$, and (ii) it respects the constraints in α, meaning that for each gate $g \in G$, if $\alpha(g)$ is defined, then $\alpha(g) = \tau(g)$. If some assignment satisfies $\mathcal{C}^\alpha$, then $\mathcal{C}^\alpha$ is *satisfiable* and otherwise *unsatisfiable*.

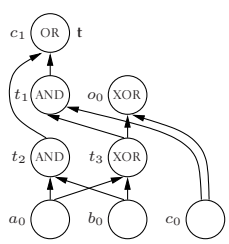
Examples of typical Boolean function applied in Boolean circuits as gate types include the following.

- $\text{NOT}(v)$ is $\mathbf{t}$ if and only if v is $\mathbf{f}$.
- $\text{OR}(v_1, \ldots, v_n)$ is $\mathbf{t}$ if and only if at least one of $v_1, \ldots, v_n$ is $\mathbf{t}$.
- $\text{AND}(v_1, \ldots, v_n)$ is $\mathbf{t}$ if and only if all $v_1, \ldots, v_n$ are $\mathbf{t}$.
- $\text{XOR}(v_1, v_2)$ is $\mathbf{t}$ if and only if exactly one of v_1, v_2 is $\mathbf{t}$.

Example 2.1 *A Boolean circuit and its graphical representation is shown in Figure 1. The circuit models a full-adder with the constraint that the carry-out bit c_1 is* $\mathbf{t}$*. One satisfying truth assignment for the circuit is*

$$\{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{t}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{f}\rangle, \langle t_3, \mathbf{t}\rangle, \langle a_0, \mathbf{t}\rangle, \langle b_0, \mathbf{f}\rangle, \langle c_0, \mathbf{t}\rangle\}.$$

The set of output gates and input gates in the circuit are $\{c_1, o_0\}$ and $\{a_0, b_0, c_0\}$, respectively.



$$\begin{aligned}\mathcal{C} \;=\; \{&c_1 := \mathrm{OR}(t_1, t_2)\\ &t_1 := \mathrm{AND}(t_3, c_0)\\ &o_0 := \mathrm{XOR}(t_3, c_0)\\ &t_2 := \mathrm{AND}(a_0, b_0)\\ &t_3 := \mathrm{XOR}(a_0, b_0)\}\end{aligned}$$

$$\alpha = \{\langle c_1, \mathbf{t}\rangle\}$$

Figure 1: A constrained Boolean circuit $\mathcal{C}^\alpha$.

The *restriction* of an assignment τ to a set $G' \subseteq G$ of gates is

$$\tau|_{G'} = \{\langle g, v\rangle \in \tau \mid g \in G'\}.$$

Given a non-input gate $g := f(g_1, \ldots, g_n)$ and a value $v \in \{\mathbf{f}, \mathbf{t}\}$, a *justification* for the pair $\langle g, v\rangle$ is a partial assignment $\sigma : \{g_1, \ldots, g_n\} \to \{\mathbf{f}, \mathbf{t}\}$ over the children of g for which $f(\tau(g_1), \ldots, \tau(g_n)) = v$ holds for all extensions $\tau \supseteq \sigma$. That is, the values assigned by σ to the children of g are enough to force g to have the value v. A gate g is *justified in an assignment* τ if it is assigned, i.e., $\tau(g)$ is defined, and

(i) g is an input gate, or

(ii) $g := f(g_1, \ldots, g_n) \in \mathcal{C}$ and $\tau|_{\{g_1,\ldots,g_n\}}$ is a justification for $\langle g, \tau(g)\rangle$.

Example 2.2 *Consider the gate t_1 in Figure 1. The set of possible justifications for $\langle t_1, \mathbf{f}\rangle$ consists of $\{\langle t_3, \mathbf{f}\rangle\}$, $\{\langle t_3, \mathbf{f}\rangle, \langle c_0, \mathbf{t}\rangle\}$, $\{\langle t_3, \mathbf{f}\rangle, \langle c_0, \mathbf{f}\rangle\}$, $\{\langle c_0, \mathbf{f}\rangle\}$, and $\{\langle t_3, \mathbf{t}\rangle, \langle c_0, \mathbf{f}\rangle\}$; the first and fourth are subset minimal ones. Gate t_1 is justified in the satisfying assignment in Example 2.1.*

Given a constrained circuit $\mathcal{C}^\alpha$ and an assignment $\tau \supseteq \alpha$ for $\mathcal{C}$, the *justification cone* of $\mathcal{C}^\alpha$ under τ, denoted by $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$, is defined recursively top-down in the circuit structure, starting from the constrained gates. Intuitively, the cone is the smallest set of gates which includes all constrained gates and, for each justified gate in the set, all the gates that participate in some subset minimal justification for the gate. More formally, $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ is the smallest one of those sets S of gates which satisfy the following properties.

1. If $\langle g, v\rangle \in \alpha$, then $g \in S$.
2. If $g \in S$ and (i) g is a non-input gate, (ii) g is justified in τ, and (iii) $\langle g_i, v_i\rangle \in \sigma$ for some subset minimal justification σ for $\langle g, \tau(g)\rangle$, then $g_i \in S$.

Notice that by this definition $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ is unambiguously defined.

The *justification frontier* of $\mathcal{C}^\alpha$ under τ is the "bottom edge" of the justification cone, that is, those gates in the cone that are not justified:

$$\mathsf{jfront}(\mathcal{C}^\alpha, \tau) = \{g \in \mathsf{jcone}(\mathcal{C}^\alpha, \tau) \mid g \text{ is not justified in } \tau\}.$$

A gate g is *interesting* in τ if it belongs to the frontier $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ or is a descendant of a gate in it. The set of all gates that are interesting in τ is

denoted by $\mathsf{interest}(\mathcal{C}^\alpha, \tau)$. A gate g is an (*observability*) *don't care* if it is neither interesting nor in the justification cone $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$.

Example 2.3 *Consider the constrained circuit $\mathcal{C}^\alpha$ in Figure 1. Under the assignment*

$$\tau = \{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{t}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{f}\rangle, \langle t_3, \mathbf{t}\rangle, \langle a_0, \mathbf{f}\rangle, \langle b_0, \mathbf{f}\rangle, \langle c_0, \mathbf{t}\rangle\},$$

the justification cone $\mathsf{jcone}(\mathcal{C}^\alpha, \tau)$ is $\{c_1, t_1, t_3, c_0\}$, the justification frontier, $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is $\{t_3\}$, $\mathsf{interest}(\mathcal{C}^\alpha, \tau) = \{t_3, a_0, b_0\}$, and the gates t_2 and o_0 are don't cares.

When $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty, a satisfying assignment can be obtained by (i) restricting τ to the input gates appearing in the justification cone, that is, to the gate set $\mathsf{jcone}(\mathcal{C}^\alpha, \tau) \cap \mathsf{inputs}(\mathcal{C})$, (ii) assigning other input gates arbitrary values, and (iii) recursively evaluating the values of non-input gates.

Proposition 2.1 *If the justification frontier, $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$, is empty for some assignment τ, then the constrained circuit $\mathcal{C}^\alpha$ is satisfiable.*

Thus, whenever $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty, we say that τ *de facto satisfies* $\mathcal{C}^\alpha$.

Example 2.4 *The assignment*

$$\tau = \{\langle c_1, \mathbf{t}\rangle, \langle t_1, \mathbf{f}\rangle, \langle o_0, \mathbf{f}\rangle, \langle t_2, \mathbf{t}\rangle, \langle t_3, \mathbf{t}\rangle, \langle a_0, \mathbf{t}\rangle, \langle b_0, \mathbf{t}\rangle, \langle c_0, \mathbf{t}\rangle\}$$

de facto satisfies the constrained circuit $\mathcal{C}^\alpha$ in Figure 1.

Also note that if a total truth assignment τ satisfies $\mathcal{C}^\alpha$, then $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty.

Note on Representation Forms for Structured Formulas

Many graph-based representation forms for propositional formulas can be found in the literature. Some of these, [149, 1, 240] for instance, can be seen as special cases of our definition for Boolean circuits. *AND-inverter graphs* (AIGs) [149] are basically Boolean circuits in which only the gate types AND and NOT are used. It should be noted that typically work on such graph presentation forms ([78, 149, 1, 15, 41] among others) deals with techniques for reducing the size of circuits in order to enable storing very large formulas. This is also the case with *reduced Boolean circuits* (RBCs), as defined in [1]. Recently, Boolean circuits have also been called *propositional DAGs* [240], although the underlying formalism coincides with Boolean circuits.

Other often studied representation forms for propositional formulas are provided by the *target languages* in *knowledge compilation*. In knowledge compilation, the basic idea is to first compile the given propositional formula into a specific target normal form in the *offline phase*. The resulting representation in the target language is then queried for results in the *online phase*. Ideally, the target language is a normal form which supports polynomial time answering for various queries, including *model counting*, *model enumeration*, and *equivalence* in addition to satisfiability. Many target compilations have been proposed, including *Horn approximations* [215] *decomposable negation normal form* (DNNF) [63] and *ordered binary decision diagrams* (OBDDs) [51, 70]. However, the fundamental weakness of

the compilation-based approaches lies in exponential worst-case space consumption, which in many cases leads to poor scalability.

Although this thesis concentrates on search-based approaches, we note that the algorithms for building (reduced) OBDDs can be seen as an inference system, which is rather different to DPLL-based methods. For studies on OBDDs as a proof system from the view point of proof complexity, see [105, 21, 145, 210].

As a final remark, *Boolean expression diagrams* (BEDs) [15] aim at extending BDDs with Boolean circuit style gate types. For a short overview of BDDs, BEDs, RBCs, and AIGs, we refer the reader to [41].

2.2 CNF SAT

Since the DPLL-based search procedures of interest in this work deal typically with CNF formulas, we will next define CNF-level SAT and a standard translation from Boolean circuits to CNF formulas.

Given a Boolean variable x, there are two *literals*, the positive literal, denoted by x, and the negative literal, denoted by $\neg x$, where $\neg$ is the logical negation (not). As usual, we identify $\neg\neg x$ with x. A *clause* is a disjunction ($\vee$, or) of distinct literals and a CNF formula is a conjunction ($\wedge$, and) of clauses. When convenient, we view a clause as a finite set of literals and a CNF formula as a finite set of clauses; for example, the formula $(a \vee \neg b) \wedge (\neg c)$ can be written as $\{\{a, \neg b\}, \{\neg c\}\}$. The sets of variables appearing as positive and negative literals in a CNF formula F are denoted by $\mathsf{vars}^+(F)$ and $\mathsf{vars}^-(F)$, respectively, and the set of variables by $\mathsf{vars}(F)$; for a clause C, $\mathsf{vars}^+(C)$, $\mathsf{vars}^-(C)$, and $\mathsf{vars}(C)$ are defined similarly.

Given a CNF formula F, a (partial) *assignment* for F is a (partial) function $\tau : \mathsf{vars}(F) \to \{\mathbf{t}, \mathbf{f}\}$, where $\mathbf{t}$ and $\mathbf{f}$ stand for *true* and *false*, respectively. With slight abuse of notation, if $\tau(x) = v$, then $\tau(\neg x) = \neg v$, where $\neg\mathbf{t} = \mathbf{f}$ and $\neg\mathbf{f} = \mathbf{t}$. A clause is *satisfied* by τ if it contains at least one literal l such that $\tau(l) = \mathbf{t}$. If $\tau(l) = \mathbf{f}$ for every literal l in a clause, the clause is *falsified* by τ. An assignment τ *satisfies* F if it satisfies every clause in it. A formula is *satisfiable* if there is a total assignment that satisfies it, and *unsatisfiable* otherwise.

Any CNF formula $F = \{C_1, \ldots, C_k\}$ can naturally be seen as a Boolean circuit. Basically, F is a Boolean circuit with an AND of ORs which represent the clauses. Formally, $\mathsf{circuit}(F) := \langle \mathcal{C}, \tau \rangle$ is defined by associating an input gate x with each variable $x \in \mathsf{vars}(F)$, a NOT-gate $g_{\neg x}$ with each $x \in \mathsf{vars}^-(F)$, an OR-gate g_{C_i} with each clause $C_i \in F$, an AND-gate g_F with F, and by setting $\tau = \{\langle g_F, \mathbf{t} \rangle\}$ and

$$\begin{aligned} \mathcal{C} \;=\; & \{g_F := \mathrm{AND}(g_{C_1}, \ldots, g_{C_k})\} \cup \{g_{\neg x} := \mathrm{NOT}(x) \mid x \in \mathsf{vars}^-(F)\} \cup \\ & \{g_{C_i} := \mathrm{OR}(\alpha(l_{i,1}), \ldots, \alpha(l_{i,n_i})) \mid C_i = \{l_{i,1}, \ldots, l_{i,n_i}\} \in F\}. \end{aligned}$$

where $\alpha(\neg x) = g_{\neg x}$ and $\alpha(x) = x$ for each $x \in \mathsf{vars}(F)$.

Example 2.5 *The constrained Boolean circuit* $\mathsf{circuit}(F)$ *for the unsatisfiable CNF formula* $F = \{\{x, y\}, \{x, \neg y\}, \{\neg x, y\}, \{\neg x, \neg y\}\}$ *is shown in Figure 2.*

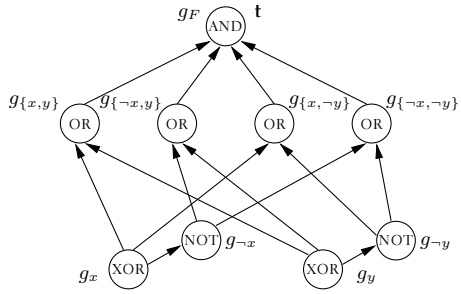


Figure 2: The constrained Boolean circuit $\mathsf{circuit}(\{\{x, y\}, \{x, \neg y\}, \{\neg x, y\}, \{\neg x, \neg y\}\})$.

2.3 Translating Boolean Circuits to CNF

In order to exploit CNF-level SAT solvers in solving instances of Boolean circuit satisfiability, the circuit has to be translated to CNF. A "standard" linear time translation based on [233] is typically applied, which achieves a linear size CNF encoding of any propositional formula by introducing additional variables for representing the subformula structure in the circuit. For encoding the functionalities of gates, the idea is to represent the logical equivalence $g \leftrightarrow f(g_1, \ldots, g_n)$ as clauses; hence for each $g := f(g_1, \ldots, g_n)$ the corresponding introduced clauses are as shown in Table 1. Similarly, a unit clause is added for each constraint $\langle g, v \rangle \in \tau$ as shown in Table 1. Given a constrained Boolean circuit $\mathcal{C}^\alpha$, we will denote its CNF translation by $\mathsf{cnf}(\mathcal{C}^\alpha)$.

Table 1: CNF translation for constrained Boolean circuits.

gate or constraint	clauses
$g := \text{XOR}(g_1, g_2)$	$(\neg\tilde{g} \vee \neg\tilde{g}_1 \vee \neg\tilde{g}_2), (\neg\tilde{g} \vee \tilde{g}_1 \vee \tilde{g}_2), (\tilde{g} \vee \neg\tilde{g}_1 \vee \tilde{g}_2), (\tilde{g} \vee \tilde{g}_1 \vee \neg\tilde{g}_2)$
$g := \text{OR}(g_1, \ldots, g_n)$	$(\neg\tilde{g} \vee \tilde{g}_1 \vee \cdots \vee \tilde{g}_n), (\tilde{g} \vee \neg\tilde{g}_1), \ldots, (\tilde{g} \vee \neg\tilde{g}_n)$
$g := \text{AND}(g_1, \ldots, g_n)$	$(\neg\tilde{g} \vee \tilde{g}_1), \ldots, (\neg\tilde{g} \vee \tilde{g}_n), (\tilde{g} \vee \neg\tilde{g}_1 \vee \cdots \vee \neg\tilde{g}_n)$
$g := \text{NOT}(g_1)$	$(\neg\tilde{g} \vee \neg\tilde{g}_1), (\tilde{g} \vee \tilde{g}_1)$
$\langle g, \mathbf{t} \rangle \in \tau$	$(\tilde{g})$
$\langle g, \mathbf{f} \rangle \in \tau$	$(\neg\tilde{g})$

It should be noted that, in practice, gates of the form $g := \text{NOT}(g_1)$ are not typically translated; instead, $\neg\tilde{g}_1$ is substituted for $\tilde{g}$.

The "standard" translation from constrained Boolean circuits to CNF formulas presented in Table 1 is often (depending on the context) referred to as "Tseitin translation", as it follows the encoding of arbitrary propositional formulas as CNF formulas presented in [233]. A well–known refinement of this standard encoding is the Plaisted–Greenbaum *polarity exploiting translation* [188]. Compact CNF encodings of Boolean circuits (or non-clausal formulas) are also developed in [67, 185, 120], for instance. However, within the scope of this work, we will apply the standard translation.

2.4 Normal Logic Programs and Stable Models

In addition to the classical notion of propositional satisfiability, we consider in publication **P5** answer set programming (ASP), or in other words, constraint

satisfaction techniques for *normal logic programs* (NLPs) (in the propositional case) under the *stable model semantics* [88, 86].

An NLP Π consists of a finite set of rules of the form

$$r \ : \ h \leftarrow a_1, \ldots, a_n, {\sim}b_1, \ldots, {\sim}b_m, \tag{1}$$

where each a_i and b_j is a propositional atom, and h is either a propositional atom, or the special symbol $\bot$ that stands for falsity. The symbol "$\sim$" denotes *default negation*. A *default literal* is a propositional atom a, or its default negation ${\sim}a$. A rule r consists of the *head*, $\mathsf{head}(r) = h$, and a *body*, $\mathsf{body}(r) = \{a_1, \ldots, a_n, {\sim}b_1, \ldots, {\sim}b_m\}$. A rule r is a *fact* if $\mathsf{body}(r) = \emptyset$.

The set of atoms occurring in a program Π is denoted by $\mathsf{atoms}(\Pi)$. We use the shorthands $L^+ = \{a \mid a \in L\}$ and $L^- = \{a \mid {\sim}a \in L\}$ for a set L of default literals, and ${\sim}A = \{{\sim}a \mid a \in A\}$ for a set A of atoms. This allows the shorthand $\mathsf{head}(r) \leftarrow \mathsf{body}(r)^+ \cup {\sim}\mathsf{body}(r)^-$ for (1). The set of default literals of a program Π is $\mathsf{dlits}(\Pi) = \{a, {\sim}a \mid a \in \mathsf{atoms}(\Pi)\}$.

In ASP, we are interested in *stable models* [88] (or *answer sets*) of an NLP Π. An *interpretation* $M \subseteq \mathsf{atoms}(\Pi)$ defines which atoms of Π are true ($a \in M$) and which are false ($a \notin M$). An interpretation $M \subseteq \mathsf{atoms}(\Pi)$ is a *(classical) model* of Π if and only if $\mathsf{body}(r)^+ \subseteq M$ and $\mathsf{body}(r)^- \cap M = \emptyset$ imply $\mathsf{head}(r) \in M$ for each rule $r \in \Pi$. A model M of a program Π is a stable model of Π if and only if there is no model $M' \subset M$ for Π^M, where

$$\Pi^M = \{\mathsf{head}(r) \leftarrow \mathsf{body}(r)^+ \mid r \in \Pi \text{ and } \mathsf{body}(r)^- \cap M = \emptyset\}$$

is called the *Gelfond-Lifschitz reduct* of Π with respect to M. We say that a program Π is *satisfiable* if it has a stable model, and *unsatisfiable* otherwise.

In the light of **P5**, an important class of NLPs are so called *tight* NLPs, as defined next. The *positive dependency graph* of Π, denoted by $\mathsf{Dep}^+(\Pi)$, is the directed graph with $\mathsf{atoms}(\Pi)$ and

$$\{\langle b, a\rangle \mid \exists r \in \Pi \text{ such that } b = \mathsf{head}(r) \text{ and } a \in \mathsf{body}(r)^+\}$$

as the sets of vertices and edges, respectively. An NLP Π is tight if there are no loops in its positive dependency graph $\mathsf{Dep}^+(\Pi)$.

2.5 Relationship between SAT and ASP

A part of this thesis (namely, publication **P5**) considers the relationship between SAT and ASP. For this, we now consider known translations between these paradigms.

There is a natural linear-size translation from CNF formulas to normal logic programs so that the stable models of the encoding represent the satisfying truth assignments of the original CNF formula *faithfully*, that is, there is a bijective correspondence between the satisfying truth assignments and stable models of the translation [182]. Given a CNF formula F, this translation $\mathsf{nlp}(F)$ introduces a new atom c for each clause $C \in F$, and atoms a_x and $\hat{a}_x$ for each variable $x \in \mathsf{vars}(F)$. The resulting NLP is then

$$\begin{aligned}
\mathsf{nlp}(F) \ = \ & \{a_x \leftarrow {\sim}\hat{a}_x.\ \ \hat{a}_x \leftarrow {\sim}a_x \mid x \in \mathsf{vars}(F)\} \cup && (2)\\
& \{\bot \leftarrow {\sim}c \mid C \in F\} \cup && (3)\\
& \{c \leftarrow a_x \mid x \in C,\ C \in F,\ x \in \mathsf{vars}(C)\} \cup && (4)\\
& \{c \leftarrow {\sim}a_x \mid \neg x \in C,\ C \in F,\ x \in \mathsf{vars}(C)\}. && (5)
\end{aligned}$$

The rules (2) encode that each variable must be assigned an unambiguous truth value, the rules in (3) that each clause in F must be satisfied, while (4) and (5) encode that each clause is satisfied if at least one of its literals is satisfied.

Example 2.6 *The CNF formula* $F = \{\{x, y\}, \{\neg x, y\}, \{x, \neg y\}, \{\neg x, \neg y\}\}$ *is represented by the normal logic program*

$$\begin{aligned}\mathsf{nlp}(F) \;=\; \{\; & a_x \leftarrow {\sim}\hat{a}_x.\ \hat{a}_x \leftarrow {\sim}a_x.\ a_y \leftarrow {\sim}\hat{a}_y.\ \hat{a}_y \leftarrow {\sim}a_y. \\ & \bot \leftarrow {\sim}c_1.\ \bot \leftarrow {\sim}c_2.\ \bot \leftarrow {\sim}c_3.\ \bot \leftarrow {\sim}c_4. \\ & c_1 \leftarrow a_x.\ c_1 \leftarrow a_y.\ c_2 \leftarrow a_x.\ c_2 \leftarrow {\sim}a_y. \\ & c_3 \leftarrow {\sim}a_x.\ c_3 \leftarrow a_y.\ c_4 \leftarrow {\sim}a_x.\ c_4 \leftarrow {\sim}a_y\ \}.\end{aligned}$$

Contrary to the case of translating SAT into ASP, there is no modular[1] and faithful translation from normal logic programs to propositional logic [182]. Moreover, any faithful translation is potentially of exponential size when additional variables are not allowed [163][2].

For any tight program Π, however, the answer sets of Π faithfully coincide with the satisfying truth assignments of a linear-size propositional formula called *Clark's completion* [56, 74] of Π. Taking a Boolean variable x_a for each $a \in \mathsf{atoms}(\Pi)$, Clark's completion is

$$C(\Pi) \;=\; \bigwedge_{h \in \mathsf{atoms}(\Pi) \cup \{\bot\}} \left(x_h \leftrightarrow \bigvee_{r \in \mathsf{rules}(h)} \left(\bigwedge_{b \in \mathsf{body}(r)^+} x_b \wedge \bigwedge_{b \in \mathsf{body}(r)^-} \bar{x}_b \right) \right),$$

where $\mathsf{rules}(h) = \{r \mid \mathsf{head}(r) = h\}$. Notice that there are the special cases that (i) if h is $\bot$, then the equivalence becomes the negation of the right hand side, (ii) if h is a fact, then the equivalence reduces to the clause $\{x_h\}$, and (iii) if h does not appear in the head of any rule, then the equivalence reduces to the clause $\{\bar{x}_h\}$.

A linear-size CNF translation of $C(\Pi)$, referred to here as the *clausal completion* $\mathsf{comp}(\Pi)$, is achieved by encoding $C(\Pi)$ in the style of the Tseitin translation, through introducing a new Boolean variable x_B for each $B \in \mathsf{body}(\Pi)$; we refer the reader to Section 2 of **P5** for details.

[1] Intuitively, for a modular translation, adding an atom as a fact to a program leads to a local change not involving the translation of the rest of the program [182].

[2] However, polynomial size propositional encodings using extra variables are known, see [35, 121]. Also, ASP as Propositional Satisfiability approaches for solving normal logic programs have been developed, for example, ASSAT [164] (based on incrementally adding—possible exponentially many—loop formulas) and ASP-SAT [89] (based on generating a *supported model* [48] of the program and testing its minimality—thus avoiding exponential space consumption).

3 PROOF SYSTEMS AND SOLVER TECHNIQUES FOR SAT

This chapter concentrates on introducing the propositional proof systems considered in this work, including DPLL and clause learning, and their circuit-level counterparts. Moreover, Section 3.6 provides a short overview of the basic ideas behind typical CNF-level stochastic local search SAT solvers. First of all, though, we will discuss proof complexity and polynomial simulation, which provide means of comparing the relative efficiency of proof systems.

3.1 Propositional Proof Systems and Complexity

Proof complexity theory enables the study of relative efficiency of solver techniques by investigating whether the proof systems underlying different solvers can *(polynomially) simulate* [60] one another. From the practical point of view, if proof system S' cannot simulate system S, any implementation of S' can suffer a notable decrease in efficiency compared to implementations of S. Due to this strong interplay between theory and practice, the study of the relative efficiency of proof systems reveals important new explanations for the successes and failures of particular solver techniques.

Formally, a *propositional proof system* [60] is a polynomial-time computable predicate S such that a propositional formula F is unsatisfiable if and only if there is a *proof* p for which $S(F, p)$ holds. Thus a proof p of F is a *certificate* of the unsatisfiability of F, and a proof system is a polynomial-time procedure for checking the validity of proofs in a certain format.

While proof checking is efficient, finding short proofs may be difficult, or, generally, impossible since short proofs may not exist for too weak a proof system. As a measure of hardness of proving unsatisfiability of a CNF formula F in a proof system S, the *(proof) complexity* $C_S(F)$ of F in S is the *length* of the shortest proof of F in S. For a family $\{F_n\}$ of unsatisfiable CNF formulas over an increasing number of variables, the (asymptotic) complexity of $\{F_n\}$ is measured with respect to the number of clauses in F_n.

For two proof systems, S and S', we say that S' *(polynomially) simulates* S if for all families $\{F_n\}$ of unsatisfiable formulas, $C_{S'}(F_n) \leq p(C_S(F_n))$ for all F_n, where p is a polynomial. If S simulates S' and vice versa, then S and S' are *polynomially equivalent*. If there is a family $\{F_n\}$ which witnesses the fact that S' does not polynomially simulate S, we say that $\{F_n\}$ *separates* S from S'. If S can be separated from S' and vice versa, then S and S' are *incomparable*. Notice that polynomial simulation gives a partial order for proof systems based on their relative power. We also note that polynomial simulation, as defined here, differs from the stronger notion of *p-simulation* which additionally requires that there is an efficient algorithm to convert the proof in one system to a short proof in the other system.

With these definitions, in order to show that a proof system S cannot simulate another system S', it suffices to exhibit an infinite family $\{F_n\}$ of unsatisfiable formulas over an increasing number of variables such that the minimum length proofs in S for $\{F_n\}$ are asymptotically superpolynomially longer than the minimum length proofs in S' with respect to the number of clauses in F_n.

3.2 Resolution

The well-known Resolution proof system [197] (RES) is based on the *resolution rule*. Let C, D be clauses, and x a Boolean variable. The resolution rule is

$$\frac{\{x\} \cup C \quad \{\neg x\} \cup D}{C \cup D}$$

or, in other words, we can *directly derive* $C \cup D$ from $\{x\} \cup C$ and $\{\neg x\} \cup D$ by *resolving on* x. For a given CNF formula F, a RES *derivation of a clause* C from F is a sequence of clauses $\pi = (C_1, C_2, \ldots, C_m = C)$, where each C_i, $1 \leq i \leq m$, is either

(i) a clause in F (an *initial clause*), or

(ii) directly derived with the resolution rule from two clauses C_j, C_k where $1 \leq j, k < i$ (a *derived clause*).

The *length* of π is m, the number of clauses occurring in it. A RES *proof (for the unsatisfiability)* of a CNF formula F is any RES derivation of the empty clause $\emptyset$ from F.

Any RES derivation $\pi = (C_1, C_2, \ldots, C_m)$ can be presented as a directed acyclic graph in which the leafs are initial clauses and other nodes represent derived clause. The edge relation is defined so that there are edges from C_i and C_j to C_k, if and only if C_k has been directly derived from C_i and C_j using the resolution rule. Many *refinements of* Resolution, in which the structure of RES proofs is restricted, have been proposed and studied. Here of particular interest is *Tree-like Resolution* (T-RES), with the requirement that proofs are representable as trees. This implies that a derived clause, if subsequently used multiple times in the proof, must be derived anew each time starting from initial clauses. Other well-known refinements include *regular resolution* [233] (any variable can be resolved upon at most once along any path in the proof from an initial clause to $\emptyset$), *Davis–Putnam (*or *ordered) resolution* [65] (a refinement of regular resolution where every sequence of variables resolved on in a path from an initial clause to $\emptyset$ respects the same ordering on the variable).

Super-polynomial (and even exponential) lower bounds on proof lengths in RES have been shown for various families of CNF formulas, see [55, 107, 233, 234, 62, 6, 29, 31] for examples. It is also known that T-RES cannot polynomially simulate RES. This originates from the facts that *regular resolution* cannot simulate RES [95, 8], and T-RES in turn cannot simulate regular resolution [235].

3.3 The Davis–Putnam–Logemann–Loveland Procedure

Given a CNF formula F as input, DPLL (as in Davis–Putnam–Logemann–Loveland [65, 64]) is a depth-first search procedure building a partial assignment τ for the variables in F through (i) *branching* and (ii) *unit propagation*. In branching, the current assignment τ is extended with the assignment (*decision*) $\langle x, v \rangle$, where v is either $\mathbf{f}$ of $\mathbf{t}$, for some unassigned variable x. Unit

propagation refers to applying the *unit clause rule*. The unit clause rules states that if there is a clause $\{l_1, \ldots, l_k, l\} \in F$ such that $\tau(l_i) = \mathbf{f}$ for each $1 \leq i \leq k$, the current partial assignment τ can be extended with $\langle l, \mathbf{t} \rangle$.

An assignment is extended until (i) some variable x would be assigned both $\mathbf{f}$ and $\mathbf{t}$ (a *conflict* is reached, with x as the *conflict variable*) or (ii) τ satisfies F (in which case DPLL terminates). In case (i), non-clause learning DPLL solvers *backtrack* to the last branching decision which has not been backtracked upon, undoing all assignments made by unit propagation after the particular decision, and flip the decision. DPLL terminates on an unsatisfiable CNF formula when there are no untried branches left.

From the proof theoretic point of view, search trees traversed by DPLL algorithms correspond to tableaux in the form of binary trees, which are build using two rules: the *branching rule* and the unit clause rule. The branching rule, corresponding to branching on a variable x, extends a branch into two branches, one of which is extended with the entry x and the other with $\neg x$. The unit clause rule is similarly applied by extending the branch with l. Starting from the clauses in the input CNF formula, a branch is (fully) extended until we have both of the entries x and $\neg x$ for some variable, or no new entries can be generated with the branching and unit clause rules. From an algorithmic point of view, the choice of in which order branches are extended is part of the solver strategy, and based on a *branching* (or *decision*) *heuristic*. The other branch resulting from the particular application of the branching rule is handled through backtracking. With this intuition, a DPLL *proof* refers to such a tableau proof build using the branching and unit clause rules. The length of a DPLL proof is defined as the number of applications of the branching rule in the proof.

One-step lookahead (see [158], for example) is an often implemented technique in (non-clause learning) DPLL algorithms. In one-step lookahead, if there is an assignment v to a currently unassigned variable x such that the current assignment τ with the addition of $\langle x, v \rangle$ leads to a conflict using unit propagation, then x is immediately assigned the value $\neg v$. This technique does not add to the strength of DPLL, since the same effect can obviously be accomplished by branching on x.

It is well-known that DPLL and T-RES can polynomially simulate each other (see [32] for example). One can show that for any unsatisfiable CNF formula, a minimum length DPLL proof, with applications of the unit clause rule "simulated by branching", always corresponds one-to-one with a minimum length T-RES proof, and vice versa.

Implication Graphs

Implication graphs capture the ways of deriving values for variables with the unit clause rule from assignments made by branching. We will apply this concept in the following for defining clause learning. However, first we need some additional terminology.

A *stage* of DPLL on a CNF formula F is characterized by the *decision literals* in the branch. Considering an arbitrary branch, the variables assigned by branching are called *decision variables* and those assigned values by unit propagation are *implied variables*, with analogous definitions for *decision literals* and *implied literals*. The *decision level of a decision variable* x is one

more than the number of decision variables in the branch before branching on x. The *decision level of an implied variable* x is the number of decision variables in the branch when x is assigned a value. The decision level of DPLL at any stage is the number of decision variables in the branch.

For a given CNF formula F and a set L of literals, we denote by $F, L \vdash_{\mathrm{UP}} l$ the fact that l can be deduced from F and L by iteratively applying the unit clause rule.

Definition 3.1 *For a CNF formula F, the* implication graph $G = \langle V, E \rangle$ *at a given stage of* DPLL *with the set D of decision literals is a directed graph. The set of nodes is*

$$V = \{\Lambda\} \cup D \cup \{l \mid F, D \vdash_{\mathrm{UP}} l\},$$

where Λ is a special conflict node*, and the edge relation is*

$$\begin{aligned} E \;=\; & \{\langle \neg l_i, l \rangle \mid \{l_1, \ldots, l_k, l\} \in F \text{ and } \neg l_1, \ldots, \neg l_k \in V\} \cup \\ & \{\langle x, \Lambda \rangle, \langle \neg x, \Lambda \rangle \mid x, \neg x \in V\}. \end{aligned}$$

For a given implication graph, a variable x with both $x, \neg x \in V$ is called a *conflict variable*, and $x, \neg x$ are *conflict literals*. An implication graph contains a conflict if it contains a conflict variable; DPLL has a conflict at a given stage if the implication graph at the stage contains a conflict.

3.4 DPLL with Clause Learning and Modern SAT Solvers

There is a significant amount of reported work on boosting the efficiency of DPLL solvers, by incorporating techniques such as *intelligent branching heuristics* (see [112, 158, 180] for examples), novel *propagation mechanisms* (for example, *binary clause reasoning* [22] and *equivalence reasoning* [157, 110]), efficient propagator implementations (*watched literals* [180]), *randomization* and *restarts* [130, 101, 37], and *clause learning* [174] into DPLL. Clause learning can be regarded as an especially important progressive step in the development of SAT solvers [130, 174, 180, 97, 72]. While new propagation mechanisms, such as equivalence reasoning, have been successfully implemented into DPLL, most clause learning solvers still rely on standard *unit propagation* as the sole propagator. As for intelligent branching heuristics, while solvers without clause learning incorporate heuristics based on literal counting [112] and/or one-step lookahead [158, 109, 14], branching in clause learning solvers is driven by learning. Most clause learning solvers implement variations of—or build on top of [72, 97, 202]—the *variable state independent decaying sum* (VSIDS) heuristic [180]. The basic idea behind VSIDS is to value variables that have played an active role in reaching recent conflicts. Moreover, clause learning enables *non-chronological backtracking* (or *backjumping*). Through these ideas, the search space traversal in modern SAT solvers is guided tightly by clause learning, with the help of unit propagation and restarts. Hence, as noted for example in [117], clause learning SAT solvers differ notably from implementations of the basic DPLL.

In more detail, clause learning DPLL algorithms differ from non-clause learning DPLL most notably in what is done when reaching a conflict. If

a conflict is reached without any branching, DPLL (with or without clause learning) determines the formula F unsatisfiable. In other cases, non-clause learning DPLL algorithms perform simple backtracking as explained in the previous section. In clause learning DPLL algorithms, however, the conflict is *analyzed*, and a *learned clause* (or *conflict clause*), which describes the "cause" of the conflict, is added to F. After this the search is continued typically by applying *non-chronological backtracking* (or *conflict-driven backjumping*) for backtracking to an earlier decision level that "caused" the conflict. Conflict-driven backjumping results in the fact that, as opposed to the basic backtracking in DPLL, the other branch (opposite value) of decision variables is not necessarily forced systematically when backtracking. In other words, branching in clause learning DPLL is seen simply as assigning values to unassigned variables, rather than as a branching rule in which by branching on a variable x the current branch is always extended into two branches, one with x and the other with $\neg x$.

Conflict Graphs and Conflict Analysis

Similarly as with DPLL, the *stage* of a clause learning DPLL algorithm is characterized by the set of decision literals. At a given stage of a clause learning DPLL algorithm, a clause is called *known* if it either appears in the original CNF formula F or has been learned earlier during the search. Conflict analysis is based on a *conflict graph*, which captures one way of reaching the conflict at hand from the decision variables by using the unit clause rule on known clauses.

Definition 3.2 *Given an implication graph G, a* conflict graph $H = (V, E)$ *based on G is any acyclic subgraph of G having the following properties.*

1. *H contains Λ and exactly one conflict literal pair $x, \neg x$.*
2. *There is a path from every node in H to Λ.*
3. *Every node $l \in V \setminus \{\Lambda\}$ either (i) corresponds to a decision literal, or (ii) has precisely the nodes $\neg l_1, \neg l_2, \ldots, \neg l_k$ as predecessors, where $\{l_1, l_2, \ldots, l_k, l\}$ is a known clause.*

A conflict graph describes a single conflict and contains only decision and implied literals that can be used in reaching the conflict when applying the unit clause rule *in some order*. Hence the way of implementing unit propagation in a solver has an effect on the choice of the conflict graph. The acyclicity of conflict graphs results from the fact that unit propagation is not used to re-derive already assigned literals.

Conflict clauses are associated with *cuts* in a conflict graph. Fix a conflict graph contained in an implication graph with a conflict. A *conflict cut* is any cut in the conflict graph with all the decision variables on one side (the *reason side*) and, in addition to Λ, at least one conflict literal on the other side (the *conflict side*). Those nodes on the reason side with at least one edge going to the conflict side in a conflict cut form a cause of the conflict. With the associated literals set to **t**, unit propagation can arrive at the conflict at hand. The disjunction of the negations of these literals is the *conflict clause associated with the conflict cut*. The strategy for fixing a conflict cut is called

the *learning scheme*. A learning scheme which always learns a currently unknown clause is called *non-redundant*.

Example 3.1 *A hypothetical conflict graph is illustrated in Figure 3. Decision literals are represented with filled circles, and implied literals with hollow circles. The decision level d of each literal l is presented with the label $l@d$. For example, the conflict variable x_{13} is at decision level 5. Notice that since the literals at decision level 4 are missing from this conflict graph, they are not part of the reason for the particular conflict. In the figure, two possible conflict cuts are shown with the associated conflict clauses.*

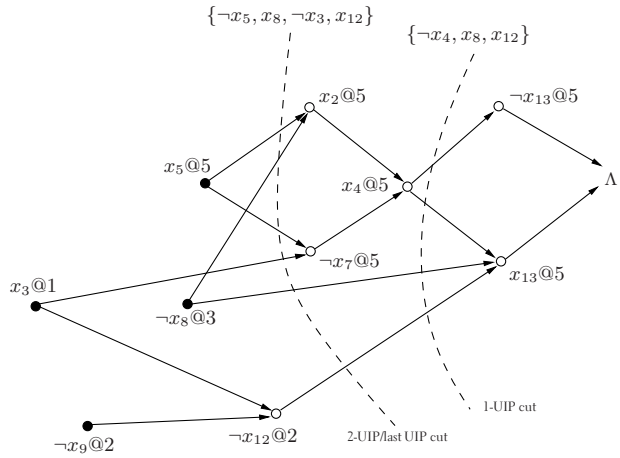


Figure 3: Example of a conflict graph, and two possible conflict cuts

Implication Points, Conflict-Driven Backjumping, and CL
Typically implemented clause learning schemes are based on *unique implication points* (UIPs) [174]. A UIP in a conflict graph is a node u on the maximal decision level d such that all paths from the decision variable x at level d to Λ go through u. Such a u always exists as x satisfies this condition. Intuitively, u is a *single* reason for the conflict at level d. Thus one can always choose a conflict cut that results in a conflict clause with a UIP as the only variable from the maximal decision level. Such a conflict clause has the property that the UIP variable can be immediately set to the value opposite to the current assignment using the unit clause rule when backtracking (the phrase "the UIP is *asserted*" is sometimes used). Furthermore, UIP learning schemes enable conflict-driven backjumping, in which DPLL backtracks to the maximal decision level of the variables other than the UIP in a conflict clause. A popular version of UIP learning is the 1-UIP scheme, where a conflict cut is chosen so that the UIP closest to Λ will be in the associated conflict clause. Different learning schemes are evaluated in [244], showing the robustness of the 1-UIP scheme in practice.

Example 3.2 *Recall the conflict graph in Figure 3. The 1-UIP in this graph is the literal x_4. One conflict cut corresponding to the 1-UIP learning scheme is the cut labeled "1-UIP cut". The cut labeled "2-UIP cut/last UIP cut" can*

result from applying the second UIP scheme *in which a conflict clause with the UIP second closest to* Λ *is chosen. In this example, the "2-UIP cut/last UIP cut" is at the same time a cut that can result from applying the* last UIP scheme *in which a cut with the decision literal on the maximal decision level as the UIP is chosen.*

For investigating the efficiency of clause learning DPLL in proof complexity theoretic terms, we need to have a proof system characterization of clause learning DPLL algorithms. We will use the following characterization, referred to as the CL *proof system*. Here we loosely follow the characterization of [33]. A clause learning proof (or CL proof) induced by a learning scheme S is constructed by applying branching and the unit clause rule, using S to learn conflict clauses when conflicts are reached, so that in the end, a *conflict can be reached at decision level zero*. When a conflict cut with a UIP is selected, it is possible to apply conflict-driven backjumping based on the conflict clause. Otherwise, simple backtracking is applied. Notice that this definition allows even the most general *nondeterministic learning scheme* [33], in which the conflict cut is selected nondeterministically from the set of all possible conflict cuts related to the conflict graph at hand.

Hence, a CL proof can be seen as a tree in which the traversal order is marked in the nodes. Each leaf node in the tree is labeled with a conflict graph, a conflict cut in the graph, and the decision level onto which to backjump. Now, the proof system CL consists of CL proofs under any learning scheme. The length of a CL proof is the number of branching decisions.

While the practical efficiency gains of implementing clause learning into DPLL-based algorithms are well-established, the first formal study on the power of clause learning is [33]: CL can provide exponentially shorter proofs than T-RES even if no restarts are allowed. Thus DPLL cannot polynomially simulate CL.

Restarts and the CL– Proof System

Restarting is an additional technique often implemented in modern solvers. When a restart occurs, the decisions and unit propagations made so far are undone, and the search continues from decision level zero. The clauses learned so far remain known after the restart. Intuitively, restarts help in escaping from getting stuck in hard-to-prove subformulas. In practice, the choice of when and how often to restart is part of the strategy of a solver. When any number of restarts are allowed during search, we say that CL has *unlimited restarts*. For a recent investigation into the effect of restarts on the efficiency of clause learning DPLL algorithms, see [117].

Beame et al. [33] define CL– as CL with branching allowed also on already assigned values. Although being non-typical in practice, this enables creating immediate conflicts at will. Although it is not known whether CL can simulate RES, it has been shown that this is true for CL– using unlimited restarts.

Theorem 3.1 ([33]) RES *and* CL– *with unlimited restarts and any non-redundant learning scheme are polynomially equivalent.*

We note that the proof of this theorem in [33] relies on the fact that unit prop-

agation is seen as applications of the unit clause rule, and hence the rule can also be left unapplied when convenient. This is non-typical for implementations of clause learning DPLL; they usually apply unit propagation eagerly whenever possible.

3.5 Circuit-Level DPLL and CL

A key element especially in publications **P3** and **P4** is the tight correspondence between a constrained Boolean circuit $\mathcal{C}^\alpha$ and its CNF translation $\mathsf{cnf}(\mathcal{C}^\alpha)$. We will now review details on the correspondence of deduction in the CNF translation of a Boolean circuit with the original circuit structure, and on how branching in DPLL and CL can be restricted based on the original circuit structure.

As there is a one-to-one relationship between the gates in a constrained Boolean circuit $\mathcal{C}^\alpha$ and the variables in the corresponding CNF formula $\mathsf{cnf}(\mathcal{C}^\alpha)$, the variables can be thought to inherit the structural properties of the gates. For example, an *input variable* is a variable that corresponds to an input gate in the original Boolean circuit, and we will take the liberty of using the terms "gate" and "variable" synonymously. Furthermore, since the CNF translation in Table 1 encodes in a natural way the semantics of the gates, unit propagation in the CNF formula can be seen as working on the level of the circuit. A further discussion on this can be found for example in [125], using a unit propagation equivalent characterization of Boolean constraint propagation as deduction rules for circuits [132]. Basically, such circuit-level Boolean constraint propagation can set a value on a gate if and only if unit propagation can set a value on the corresponding Boolean variable in the CNF translation. For example, consider the gate $g := \mathrm{AND}(g_1, g_2)$ and its CNF translation $(\neg\tilde{g} \vee \tilde{g}_1) \wedge (\neg\tilde{g} \vee \tilde{g}_2) \wedge (\tilde{g} \vee \neg\tilde{g}_1 \vee \neg\tilde{g}_2)$. Now whenever the gate g_2 is assigned to **f**, the gate g can be propagated to **f** by the semantics of AND. On the CNF-level, we can equivalently propagate the variable $\tilde{g}$ to **f** by applying the unit clause rule whenever the variable $\tilde{g}_2$ is assigned to **f**. The same kind of equivalent behavior is noticed in a "top-down" fashion when assigning the gate g to **t**: on the circuit-level, the gates g_1 and g_2 can be propagated to **t**, and on the CNF-level we can equivalently propagate the variables $\tilde{g}_1$ and $\tilde{g}_2$ to **t** through the clauses $(\neg\tilde{g} \vee \tilde{g}_1)$ and $(\neg\tilde{g} \vee \tilde{g}_2)$, respectively, by applying the unit clause rule whenever the variable $\tilde{g}$ is assigned to **t**. Hence we will also take the liberty of saying that unit propagation sets a value on a gate when referring to unit propagation setting a value on the corresponding Boolean variable in the CNF translation. Similarly, we *branch on a gate* when referring to branching on the corresponding Boolean variable. Correspondingly, a DPLL or CL proof of a constrained circuit $\mathcal{C}^\alpha$ means a proof of the translation $\mathsf{cnf}(\mathcal{C}^\alpha)$.

Since unit propagation can be also seen as Boolean constraint propagation on the level of constrained circuits, DPLL can also be implemented as a circuit-level procedure, see, for example, [172, 132, 149, 230]. Since conflict graphs are based on how the unit clause rule is applied, clause learning can also be incorporated in such circuit-level DPLL-based solvers [149, 230]. Thus the results in this thesis (in publications **P3** and **P4**) concerning the relative power of restricted branching variants of DPLL and CL hold for such

circuit-level approaches, too.

3.6 Stochastic Local Search Procedures for SAT

In addition to considering DPLL-based complete methods for SAT, in this thesis (namely, in **P6** and **P7**) we develop novel stochastic local search (SLS) techniques aimed specifically at solving structured SAT instances.

There is a wide body of work concentrating on the development of effective SLS methods for SAT (see [218, 213, 217, 177, 115, 114, 212, 111] for examples, and [116] for a general view). Such methods are typically based on iterating over solution candidates by *flipping* value assignments (inverting the assignment on a single variable) in the current candidate based on a *cost function*. In this work, we will compare the novel SLS method we develop with typical CNF-level SLS methods belonging to the *WalkSAT family*.

The WalkSAT family of SLS methods builds on top of a greedy local search method for SAT called GSAT [218]. Given a CNF formula F, GSAT searches for a satisfying truth assignment starting from a randomly chosen truth assignment over all the variables in F. One step of GSAT consists of a *move* from the current truth assignment (*configuration*) τ to another assignment τ' by flipping the assignment of a single variable in τ. The heuristics for choosing the greedy move (that is, which variable to flip) is based on the so called *break-count*, $\textsc{BreakCount}(F, \tau, x)$. For each variable $x \in \mathsf{vars}(F)$, the value of $\textsc{BreakCount}(F, \tau, x)$ is the total number of clauses not satisfied by the configuration τ' which results from flipping the value of x in the current configuration τ. In GSAT, the variable to be flipped is randomly chosen from those variables in F for which the value of $\textsc{BreakCount}(F, \tau, \cdot)$ is minimal.

The number of moves is limited to a pre-defined number $\textsc{MaxMoves}$, constituting a *try* of the algorithm. The number of tries is bounded by the value $\textsc{MaxTries}$.

The methods in the WalkSAT family of SLS methods fall into the generic WalkSAT procedure presented as Algorithm 1. Compared to GSAT, the main novel ideas in the WalkSAT family of methods are *focused moves* and the introduction of *noise* to the search. Moves are focused on the variables that occur in those clauses in F which are not satisfied by τ. One move consists of flipping a variable that occurs in a randomly chosen unsatisfied

Algorithm 1 WalkSAT

Input: CNF formula F, noise parameter $p \in [0, 1]$
Output: a satisfying assignment for F or "*don't know*"

1: **for** $try := 1$ to $\textsc{MaxTries}$ **do**
2: $\quad\tau :=$ a random truth assignment over all variables in F
3: $\quad$**for** $move := 1$ to $\textsc{MaxMoves}$ **do**
4: $\quad\quad$**if** τ satisfies F **then return** τ
5: $\quad\quad C :=$ a randomly chosen clause not satisfied by τ
6: $\quad\quad v :=$ a variable chosen using heuristic $\textsc{Select}(F, C, \tau)$
7: $\quad\quad\tau := \tau$ with the assignment on v flipped
8: **return** "*don't know*"

clause C. In a greedy move, a variable in C is flipped for which the value of $\textsc{BreakCount}(F, \tau, x)$ is minimal over the variables in C. In addition to greedy moves, *non-greedy* moves are made, bringing more randomization into the search. In a non-greedy move, a randomly chosen variable occurring in the clause C is flipped. The amount of randomization is dictated by the *noise parameter* $p \in [0, 1]$, that is, non-greedy moves are done with probability p. Intuitively, the introduced noise helps the search in escaping from local minima.

The various proposed variants of WalkSAT differ in how the heuristic $\textsc{Select}(F, C, \tau)$ is defined. We will now present some proposed variants of $\textsc{Select}(F, C, \tau)$. The first one is the original WalkSAT heuristic, WalkSAT/SKC as defined in [216], which we will in the following refer to simply as WalkSAT.

WalkSAT/SKC [216]

> If there is a variable in $\mathsf{vars}(C)$ which can be flipped without making any currently satisfied clause unsatisfied, then randomly choose one such variable and flip it. Otherwise do the following. With probability $(1-p)$, select the variable to be flipped from those variables in $\mathsf{vars}(C)$ for which the value of $\textsc{BreakCount}(F, \tau, \cdot)$ is minimal over the variables in $\mathsf{vars}(C)$ (the greedy move), or, with probability p, select the variable to be flipped randomly from the variables in $\mathsf{vars}(C)$.

In addition to using WalkSAT in the experiments in **P6**, we will use in **P7** a variant of the Novelty+ heuristic, AdaptNovelty+, which uses an adaptive mechanism to adjust the noise parameter p *during search* [114]. The Novelty+ is itself based on the Novelty heuristic [177], as defined next.

Novelty [177]

> If there is a variable $x \in \mathsf{vars}(C)$ for which it holds that (i) the value $\textsc{BreakCount}(F, \tau, x)$ is minimal over the variables in $\mathsf{vars}(C)$, and (ii) x was not flipped in the previous move, then flip x. Otherwise do the following. With probability $(1 - p)$, flip x, or, with probability p, flip the variable in $\mathsf{vars}(C)$ with the second-smallest value of $\textsc{BreakCount}(F, \tau, \cdot)$ over the variables in $\mathsf{vars}(C)$. In each case, ties are broken by choosing the least recently flipped variable.

Novelty+ [113]

> By introducing an additional parameter $wp \in [0, 1]$ (set to 0.01 in [113]), do the following. With probability wp, select the variable to be flipped randomly from the variables in $\mathsf{vars}(C)$, or, with probability $(1 - wp)$, use the Novelty heuristic.

However, we will postpone the details of the adaptive noise mechanism in AdaptNovelty+ to Section 4.5, where we develop such mechanisms specifically for the novel SLS method introduced in this thesis.

4 OVERVIEW OF MAIN RESULTS

In this section a technical overview of the main results of publications **P1**–**P7** is presented. Typically, proofs and related constructions are omitted when discussing theoretical contributions. Similarly, only selected experimental results are presented in detail.

The publications divide into topics as follows.

- Publication **P1** addresses the problem of generating hard satisfiable CNF SAT instances by introducing the regular XORSAT model. Publication **P1** is discussed in Section 4.1.
- Publications **P2**–**P4** deal with the effects of structure-based branching restrictions on DPLL-based SAT solvers:
  - Publication **P2** presents an extensive experimental evaluation of the effect of static branching restriction on clause learning SAT solving.
  - Publications **P3** and **P4** address the effect of branching restrictions on the proof complexity theoretic power of DPLL and CL.

  The main results of **P2** are discussed in Section 4.2, and the results of **P3** and **P4** in Section 4.3.
- Publication **P5** introduces the *Extended ASP Tableaux* proof system in the context of answer set programming. The main results of **P5** are discussed in Section 4.4.
- Publications **P6** and **P7** develop structure-based heuristics for solving SAT by local search, by introducing the justification-based local search method BC SLS. The main results of **P6** and **P7** are discussed in Section 4.5.

4.1 P1: Hard Structured Satisfiable CNF SAT Instances

From the perspective of solver development, benchmarks that are empirically hard for a particular class of SAT solvers reveal the practical gains of new search techniques. Benchmark sets arising from industrial applications constitute perhaps the most relevant test cases from a practical perspective. However, they have some limitations especially in early stages of solver development. Industrial benchmarks typically come only as individual instances that are quite large, which makes them difficult to use in testing new algorithmic ideas, as large instances require substantial effort on optimizing key routines in the solver. Moreover, benchmarks consisting of a limited number of individual instances are not well suited for measuring improvement and scalability.

The constructions for hard CNF families applied in proof complexity theoretic analysis exploit instances with specific structure [233, 107, 234, 62, 6]. An example of the practical implications of such constructions comes from the fact that often families of instances with a high number of *symmetries* are considered, based on, for example, the so called *pigeon-hole principle* [107]. This has motivated the development of specialized symmetry exploiting techniques applied before (see [11], for instance) and during search [159, 203]. In theoretical terms, such techniques bring solvers closer to the power of *symmetric resolution* [146, 236].

From the basic proof complexity theoretic perspective, however, only *unsatisfiable* formulas (and hence proofs of unsatisfiability) are of interest. Although it has been shown that with certain branching heuristics DPLL search takes exponential time with high probability on families of *satisfiable* formulas [184, 3, 2, 7], a satisfying truth assignment acts as a polynomial length witness for the satisfiability of any satisfiable formula.

In many applications the instances of interest are in fact satisfiable and the key task of a SAT solver is to find a satisfying truth assignment [40, 57, 140, 151]. Satisfiable benchmarks are also of interest for comparing different heuristics in their ability to guide the search towards a satisfying truth assignment. Moreover, only satisfiable instances are relevant for benchmarking incomplete local search methods.

Publication **P1** considers the problem of generating CNF SAT instances that are guaranteed to be satisfiable and experimentally hard for both DPLL-based and SLS SAT solvers. Namely, a family of satisfiable CNF formulas—(*random*) *Regular XORSAT*—is developed and studied by transforming *random regular graphs* into systems of linear equations modulo 2 presented in CNF. Additionally, techniques for hiding the underlying linearity of XORSAT instances are developed. These techniques enable transforming XORSAT-style CNF SAT instances into interesting benchmarks especially for evaluating equivalence reasoning techniques incorporated in SAT solvers.

The Regular XORSAT Construction

Let n be the number of variables in the regular XORSAT instance to be constructed. Let $X = \{x_0, x_1, \ldots, x_{n-1}\}$ be an associated set of n Boolean variables and let $Y = \{y_0, y_1, \ldots, y_{n-1}\}$ be a set of n elements, each corresponding to an equation in a system of n equations over X. A *constraint*

graph $G = (V, E)$ with bipartition (X, Y) characterizes the occurrences of the variables in the equations, that is, $\{x_j, y_i\}$ is an edge of G if and only if the variable $x_j \in X$ occurs in the equation $y_i \in Y$. The foundation of the random regular XORSAT construction lies in selecting a constraint graph G uniformly at random from the set of all 3-regular graphs with bipartition (X, Y). As a running example for this section, consider the graph in Figure 4.

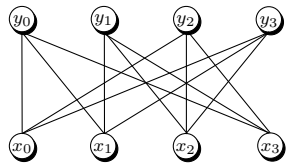


Figure 4: A 3-regular constraint graph

Once a constraint graph G has been selected, we construct a system of linear equations based on G as follows. Let $A = (a_{ij})$ be the $n \times n$ matrix whose entries are defined for all $i, j = 0, 1, \ldots, n-1$ by

$$a_{ij} = \begin{cases} 1 & \text{if } \{x_j, y_i\} \in E, \\ 0 & \text{if } \{x_j, y_i\} \notin E. \end{cases}$$

Then we select uniformly at random a $\vec{z} \in \{0,1\}^n$ and let $\vec{b} \in \{0,1\}^n$ so that $\vec{b} \equiv A\vec{z} \pmod 2$. The system of linear equations is now $A\vec{x} \equiv \vec{b} \pmod 2$, where $\vec{x} = (x_0, x_1, \ldots, x_{n-1})$ is a column vector of variables. Note that by construction $A\vec{z} \equiv \vec{b} \pmod 2$, so the system always has at least one solution—if a unique solution is required, then the matrix A must be invertible modulo 2. As an example, from the constraint graph in Figure 4 we obtain the matrix

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

For a randomly chosen vector

$$\vec{z} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad \text{we arrive at} \quad \vec{b} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

and the equation system

$$\begin{cases} x_0 + x_1 + x_3 &= 0 \\ x_1 + x_2 + x_3 &= 0 \\ x_0 + x_2 + x_3 &= 1 \\ x_0 + x_1 + x_2 &= 0 \end{cases} \pmod 2.$$

Finally, we transform the system $A\vec{x} \equiv \vec{b} \pmod 2$ into a CNF formula by introducing for every equation $x_{j_1} + x_{j_2} + x_{j_3} \equiv b_i \pmod 2$ a set of four clauses

that forbid the combinations of truth values that violate the equation. For example, the equation $x_0 + x_1 + x_2 \equiv 0 \pmod 2$ transforms into the clauses $\{\neg x_0, \neg x_1, \neg x_2\}$, $\{\neg x_0, x_1, x_2\}$, $\{x_0, \neg x_1, x_2\}$, and $\{x_0, x_1, \neg x_2\}$. For the running example, the resulting regular XORSAT instance is

$$\{\{\neg x_0, \neg x_1, \neg x_3\}, \{\neg x_0, x_1, x_3\}, \{x_0, \neg x_1, x_3\}, \{x_0, x_1, \neg x_3\},$$
$$\{\neg x_1, \neg x_2, \neg x_3\}, \{\neg x_1, x_2, x_3\}, \{x_1, \neg x_2, x_3\}, \{x_1, x_2, \neg x_3\},$$
$$\{\neg x_0, \neg x_2, x_3\}, \{\neg x_0, x_2, \neg x_3\}, \{x_0, \neg x_2, \neg x_3\}, \{x_0, x_2, x_3\},$$
$$\{\neg x_0, \neg x_1, \neg x_2\}, \{\neg x_0, x_1, x_2\}, \{x_0, \neg x_1, x_2\}, \{x_0, x_1, \neg x_2\}\}.$$

Experiments

We compare the empirical hardness of regular XORSAT to other known families for both DPLL-based and local search SAT solver. The following benchmark families are considered:

random 3-XORSAT: satisfiable random 3-XORSAT at the phase-transition point $\alpha_x = 0.918$ [193].

Jia et al's 3-XORSAT : Jia et al's generator [127] motivated by a spin glass model [181] on a rhombus with cyclic boundary conditions (satisfiable "spin glass formulas").

random 3-SAT : random 3-SAT at the phase transition point $\alpha_s = 4.27$ [61].

q-hidden: Jia et al's generator for "deceptive q-hidden" satisfiable 3-SAT formulas [128] at $q = 0.3$ and at the threshold $q = 0.618$ (q-hidden formulas).

We will compare these families to regular XORSAT which we refer to as follows:

3-regular, unique: regular XORSAT with exactly one satisfying truth assignment.

3-regular, nonunique: regular XORSAT with at least one satisfying truth assignment.

The DPLL-based solvers used are the clause learning solvers SatEliteGTI (that is, MiniSAT [72] with the SatElite preprocessor [71]) and zChaff [180], and additionally the DPLL-based lookahead solver Satz [156] which does not incorporate clause learning. Among the instance families, our generator gives the hardest instances for SatEliteGTI, zChaff and Satz; taking SatEliteGTI as an example, we have in Figure 5 for each number of variables the median number of decisions over 15 instances on a base-10 logarithmic scale. Among SatEliteGTI, zChaff and Satz, the difference between the hardness of regular XORSAT and the other considered instance families is most drastic for Satz.

In addition to DPLL-based solvers, we experiment with the SLS solvers WalkSAT [217] and AdaptNovelty+ [114]. As shown in **P1**, it appears that regular XORSAT is the hardest of the considered families for local search, too. Notice that while Survey Propagation [49] (SP) is extremely efficient in solving random k-SAT formulas, it has been observed to exhibit very poor performance on XORSAT [28]; thus we do not consider SP here.

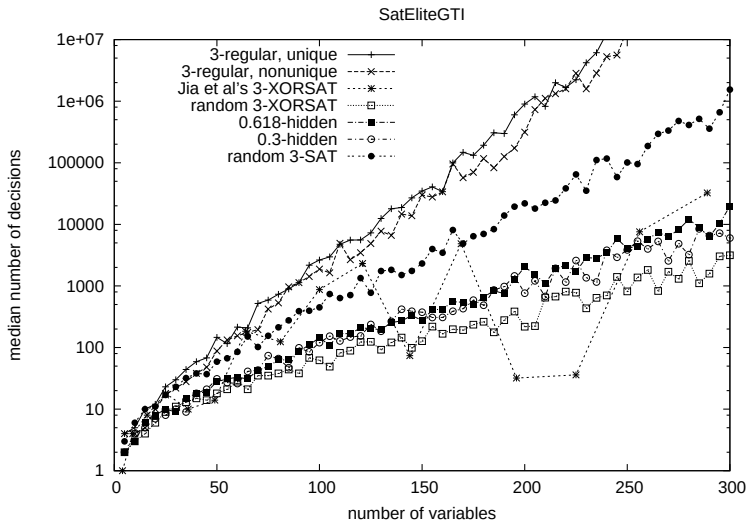


Figure 5:

Motivating the Regular XORSAT Model

The two previously proposed benchmark families most resembling regular XORSAT are the random 3-XORSAT instances [193] and the 3-XORSAT family described by Jia et al. [127] motivated by spin glass models from statistical physics. Comparing regular XORSAT and random 3-XORSAT (with forced satisfiability), the regular XORSAT construction uses a random regular constraint graph whereas in random 3-XORSAT the constraint graph is formed by associating independently and uniformly a set of three variables with each of the m equations, m being an additional parameter. Thus, given n variables, the regular XORSAT construction produces exactly n equations with exactly three occurrences of each variable, whereas in random 3-XORSAT the number of occurrences of a variable is a binomially distributed random variable with expectation $3m/n$. Comparing regular XORSAT and the spin glass 3-XORSAT family described by Jia et al. [127], in the regular XORSAT construction the regular constraint graph is selected uniformly at random whereas Jia et al. associate with each (square) n a fixed regular constraint graph derived from a $\sqrt{n} \times \sqrt{n}$ rhombic lattice with cyclic boundary.

Given this resemblance to existing benchmarks, it is not immediate why combining regularity and random selection should yield results any different compared with existing benchmarks. For example, the clauses-to-variables ratio commonly used to predict satisfiability and computational difficulty in the context of random 3-SAT [61] and random 3-XORSAT [193] is equal to 4 for both regular XORSAT and the spin glass 3-XORSAT instances: each linear equation with 3 variables is expressed with 4 clauses.

Nevertheless, the experiments show that regular XORSAT instances exhibit more rapid exponential scaling for state-of-the-art SAT solvers than the aforementioned benchmarks. Considering DPLL without clause learning, the intuition behind the difficulty of solving regular XORSAT instances comes from the fact that the instances are "highly connected". This severely

limits the effectiveness of unit propagation, as explained next.

Suggested in particular by the proofs of hardness in [7, 36, 234] (see also [33]), the connectedness of an equation system can be measured by the expansion properties of the underlying constraint graph. Here we focus on edge expansion. For a graph $G = (V, E)$ and a set $U \subseteq V$ of vertices, let $\partial_G U$ be the set of all edges in G that are incident with exactly one vertex in U. Call $\partial_G U$ the *boundary* of U. The *expansion coefficient* of G—alternatively, the *isoperimetric number* of G—is defined by

$$h(G) = \min \left\{ \frac{|\partial_G U|}{|U|} \ : \ U \subseteq V,\ 1 \leq |U| \leq \frac{|V|}{2} \right\}.$$

To provide an intuition why expansion is relevant in limiting the unit propagation, let us derive an upper bound on the number of unit propagations in terms of the expansion coefficient $h(G)$ of the constraint graph G underlying a regular XORSAT instance. Our interest is to bound the number of variables assigned by unit propagation based on the number of decisions and the expansion coefficient.

In the present context of linear equations with exactly 3 variables each, unit propagation can be seen as the application of the following rule until no more variables become determined: letting $S \subseteq X$ be the set of variables whose value has been determined so far, if the instance contains an equation with 2 variables in S and 1 variable $x_j \in X \setminus S$, then we can insert x_j into S. We denote by $\bar{S}$ the closure of S with respect to the unit clause rule. At any stage of DPLL search, the set of variables assigned by unit propagation is the unit clause rule closure on the current set of decision variables. Recall that we write n for the number of variables.

Theorem 4.1 *If $|S| \leq (n-2)h(G)/3$, then $|\bar{S}| \leq (3/(2h(G)) + 1/2)|S|$.*

Assuming that $h(G)$ has a constant nonzero lower bound as the number of variables n increases, this theorem shows that the number of variables which are assigned by unit propagation is linearly bounded by the number of current decision variables. Thus, assuming that conflicts are infrequent until a large number of variables are assigned, the theorem shows that many decision variables are required, and thus the DPLL proof will be large. Thus, hypothetically, the larger the expansion coefficient $h(G)$, the larger the proof.

To motivate our choice of random regular graphs in this light, we first observe that computing $h(G)$ for a given graph G is NP-hard [42]. However, $h(G)$ can be bounded for a d-regular graph G in terms of $\lambda_2(G)$, the second largest eigenvalue of an adjacency matrix of G, as follows [9, 10, 229]:

$$\frac{d - \lambda_2(G)}{2} \leq h(G) \leq \sqrt{2d(d - \lambda_2(G))}. \tag{6}$$

The construction of explicit infinite families of d-regular graphs with a constant nonzero lower bound on $h(G)$ is a nontrivial task; see [192] and the references therein for an account of known explicit constructions. For example, the family of 3-regular constraint graphs underlying the spin glass XORSAT instances in [127] is *not* expanding in this sense—it can be checked that the expansion coefficient has an $O(1/\sqrt{n})$ upper bound. Fortunately, most

d-regular graphs have good expansion properties [43, 143], so perhaps the easiest and most versatile way to obtain a d-regular graph with good expansion properties is to select one uniformly at random. For our present purposes we require a 3-regular graph admitting a fixed bipartition; also with this restriction it is possible to prove that most graphs admit a constant nonzero lower bound on $h(G)$. Experimentation suggests that, in practice, the standard uniform sampling procedure we use (see Section 4.1 of **P1**) produces graphs with $\lambda_2(G)$ close to $2\sqrt{2} \approx 2.8284$, which is the asymptotic optimum in terms of (6) for $\lambda_2(G)$ on 3-regular graphs [9, 167]. We note that it has also been rigorously shown that random regular graphs have near-optimal second eigenvalues [76].

Schemes for Introducing Nonlinearity

A system of linear equations modulo 2 cannot in itself be considered hard; both the existence and nonexistence of a solution can easily be determined by Gaussian elimination. However, DPLL itself does not include any special techniques for equivalence reasoning. As linear substructures often occur in real-world application domains of DPLL-based solvers (such as hardware verification), the gains from introducing equivalence reasoning into DPLL solvers seem evident. Indeed, equivalence reasoning techniques are a recent development in DPLL-based SAT solvers [109, 157, 242]. To facilitate benchmarking of equivalence reasoning techniques, we propose the following schemes for introducing nonlinearity into regular XORSAT.

Naive Scheme. Introduce three new variables x, y, z, and insert the literal x into each original clause. Additionally, add 7 clauses that force x to 0 and y, z into unique truth values.

Covering Scheme. Select a minimal set of variables such that every clause contains at least one selected variable. For each selected variable, x, introduce a new variable, y, and then substitute each occurrence of x (respectively, $\neg x$) in the clauses with $x \wedge y$ (respectively, $\neg(x \wedge y) \equiv \neg x \vee \neg y$). After all the substitutions have been performed, expand any conjuncts inside disjuncts to obtain a set of clauses.

The naive scheme is intended for benchmarking preprocessors with respect to their ability to detect the CNF representation of a set of linear equations that is conditional on a single variable x. The covering scheme is designed for benchmarking dynamic equivalence reasoning techniques that are applied during search. Ideally, a solver should be able detect and exploit the underlying linear substructure that is revealed when variables are assigned truth values during search. Additionally, extensions of these two basic schemes called *k-Nonlinear Covering Scheme*, *p-Covering Scheme*, and *p-Mixed Covering/Naive Scheme*, are provided in **P1**.

We also investigate in **P1** the effect of applying the naive and covering schemes for regular XORSAT on the efficiency of different DPLL-based solvers. In addition to SatEliteGTI, zChaff and Satz, we use march_dl [109] and EqSatz [157], both of which incorporate equivalence reasoning techniques. The experiments illustrate fundamental differences between the equivalence reasoning techniques applied in march_dl and EqSatz; for more details, we refer the reader to **P1**.

Further Developments

To keep the instance size small in relation to the number of variables, in the present study we have considered only instances in which the underlying constraint graph is d-regular with $d = 3$, resulting in 3-SAT instances. Publication **P1** mentions investigating the hardness of the Regular XORSAT instances for $d > 3$, that is, Regular d-XORSAT, as a topic for further study. After **P1** appeared, the empirical difficulty of Regular d-XORSAT has been considered by the author in [122]. As for the mentioned more theoretical topic for further work in **P1** in the form of a more rigorous analytical study of regular XORSAT, in particular in the context of local search: this question is addressed in [135], confirming the existence of high "potential barriers" (as mentioned in **P1**) based on the expansion properties of the underlying constraint graphs.

4.2 P2: Structure-Based Branching in Practice

Publication **P2** presents an extensive experimental evaluation of the effect of structure-based branching restrictions on the efficiency of solving structural SAT instances. The aim is to provide a detailed picture of the effect of branching restrictions on the inner workings of modern clause learning solvers, and to understand how important underlying structural properties of variables are in making decisions in clause learning SAT solvers. While clause learning SAT solvers typically work on the CNF-level, we derive the branching restrictions from the Boolean circuit structure underlying the CNF formulas. The motivation for the starting point of this work is that the set of input variables—when the underlying circuit structure is known—provides an easily detectable backdoor.

For the experiments we apply BCMinisat, the Boolean circuit front-end part of the BCTools [131] for the successful clause learning SAT solver Minisat [72]. The benchmark set consists of instances from a number of real-life application domains, for which Boolean circuits offer a natural representation form: verification of super-scalar processors [239], integer factorization based on hardware multiplier designs [190], equivalence checking of hardware multipliers, bounded model checking (BMC) for deadlocks in asynchronous parallel systems modeled as labeled transition systems (LTSs) [133], and linear temporal logic (LTL) BMC of finite state systems with a linear encoding [152].

We start with an overview of the effects of restricting branching in Minisat to inputs variables on the efficiency of the solver. The input-restricted branching BCMinisat is denoted by $\text{BCMinisat}_{\text{inputs}}$. After this, we will consider the effects of relaxing the input-restriction.

Effects of Input-Restricted Branching

We start with the following hypothesis.

Hypothesis 1 *The set of input variables, being a relatively small strong backdoor set, provides a branching restriction from which clause learning SAT solvers using the VSIDS heuristic benefit.*

The effect of input-restricted branching varies depending on whether unsatisfiable or satisfiable instances are considered. This is shown in the scatter plots in Figure 6, where we have the running times of the original unrestricted branching BCMinisat and the input-restricted branching $\text{BCMinisat}_{\text{inputs}}$ on the x and y axis, respectively.

On unsatisfiable instances, the input-restriction results in a clear efficiency decrease, with timed out runs shown on the horizontal line. For satisfiable instances, there seems to be no clear winner. However, $\text{BCMinisat}_{\text{inputs}}$ does timeout on several satisfiable instances, while BCMinisat timeouts only once.

We observe that, in contradiction with Hypothesis 1, the clause learning solver Minisat, with the VSIDS heuristics, shows an evident reduction in

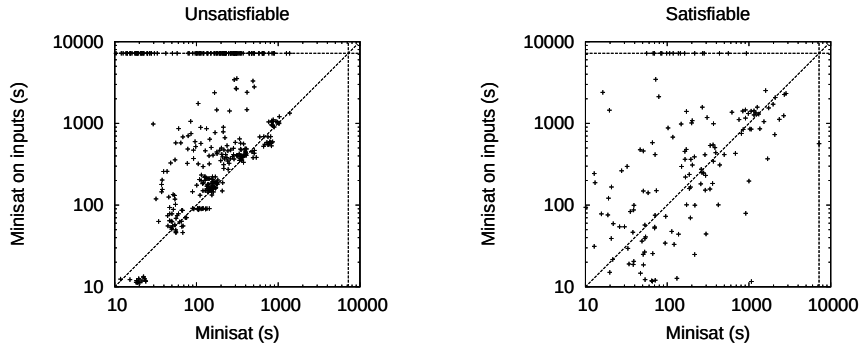


Figure 6: Comparison of BCMinisat and BCMinisat$_{\text{inputs}}$: running times on unsatisfiable (left) and satisfiable (right) instances

efficiency when restricting Minisat to branch only on input variables. Interestingly, we observe that input-restricted branching Minisat manages *fewer decision per second.*

Since the clause learning mechanism and the VSIDS heuristics, which is tightly bound with the learning mechanism, are key factors in the efficiency of Minisat, we will look for explanations for the performance of BCMinisat$_{\text{inputs}}$ by considering the effect of the input-restriction on the behavior of clause learning and VSIDS. In particular, we consider the following hypotheses.

Hypothesis 2 *By restricting branching to input variables, clause learning becomes less effective.*

Hypothesis 3 *By restricting branching to input variables, the solver is forced to make heuristically unimportant decisions.*

An important aspect in the effectiveness of clause learning is the length of conflict clauses, that is, the number of literals in the clauses. Since a conflict clause describes an unsatisfiable part of the search space, shorter conflict clauses are intuitively exponentially more effective than longer ones. In Figure 7 we have a comparison of the average lengths of conflict clauses in the solved instances. With input-restricted branching the conflict clauses are typically longer. This supports Hypothesis 2.

Further explanation for the reduced number of decisions per second and the increase in the length of conflict clauses is provided by comparing BCMinisat and BCMinisat$_{\text{inputs}}$ with respect to the number of variables assigned by unit propagation (Figure 8).

We observe that, on the average, BCMinisat$_{\text{inputs}}$ does both more propagation per decision and ventures more often into conflicts. At the same time, the conflicts BCMinisat$_{\text{inputs}}$ ventures into result in longer (and thus less effective) conflict clauses using the 1-UIP conflict learning scheme. We also

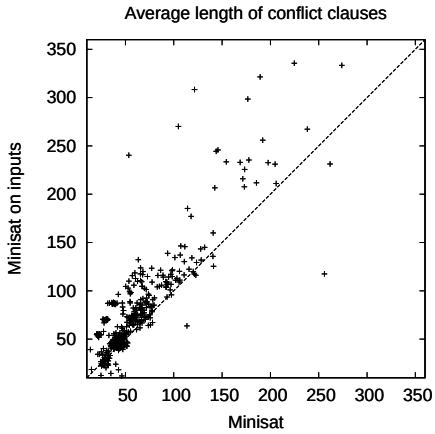


Figure 7: Comparison of BCMinisat and BCMinisat$_{\text{inputs}}$: average length of conflict clauses

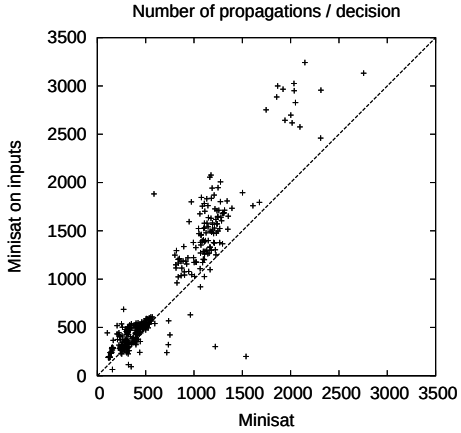


Figure 8: Comparison of BCMinisat and BCMinisat$_{\text{inputs}}$: number of propagations / decision.

observe an increase in the number of conflicts per decision. This leads us to conjecture the following. The combination of increased number of conflicts per decision and propagations per second results in a decrease in the number of decisions the solver is able to make per second. In other words, the input-restricted branching solver uses relatively more time on propagation and especially, due to the increased number on conflicts per decision, on conflict analysis. Additionally, the increase in time used for conflict analysis does not pay off, since the resulting conflict clauses are longer and thus relatively ineffective. It is very interesting to notice that an increase in the number of propagations does not seem to result in increased solver performance. This is surprising, since it is common to think that the more the solver can unit propagate, the better. It seems that the effectiveness of clause learning depends more on the *specific value assignments* that have been made rather than *how many assignments* have been made, and is in support of Hypothesis 3. This

is very much in contrast with DPLL solvers without clause learning, in which unit propagation plays an important role in pruning the search space due to standard backtracking and lack of conflict analysis.

Considering Hypothesis 3 further, we observe that the VSIDS heuristic does not seem to work as intended with input-restricted branching. The number of unbranchable variables which have better heuristic values than the best branchable variable can be high per decision. Additionally, the fraction of increments on branchable variables from the number of all increments to heuristic values during search can be in some cases even as low as 1%. Since the heuristic scores of the variables on which $\text{BCMinisat}_{\text{inputs}}$ is allowed to branch are very infrequently updated, the input-restriction results in the risk of degenerating VSIDS into a random heuristic.

The evidence supporting Hypotheses 2 and 3 leads one to question how often the original BCMinisat, without any restriction on which variables to branch on, actually branches on input variables.

Hypothesis 4 *Input variables are seldom decision variables.*

However, based on further evidence provided in **P2**, it seems that the reason for the difference in running times for unrestricted and input-restricted branching Minisat is not due to unrestricted Minisat making relatively few decisions on input variables, but rather—in disagreement with Hypothesis 4—due to the fact the the unrestricted solver can branch on other relevant variables in addition to inputs.

Based on the evidence provided this far, we conjecture that, at least without fundamentally modifying conflict learning and branching heuristics, it is unlikely that input-restricted branching can be successfully incorporated into clause learning solvers with VSIDS. The evidence against Hypothesis 4 leads us to conjecture that in order to regain robustness of the solver, the input-restriction needs to be relaxed by allowing branching on additional variables.

Relaxed Structure-Based Branching Restrictions

The conclusions on the performance degrading effects of input-restricted branching lead us to the question of how the number of variables on which the solver is allow to branch correlates with solver performance. Can the robustness of input-restricted branching be improved while still branching on a subset of variables? Another aspect is whether structural properties of the variables on which the solver is allowed to branch affect the performance of the solver.

To investigate these questions, we apply controlled schemes for allowing branching additionally on CNF variables other than input variables based on structural properties of Boolean circuits. The general idea here is to allow—in addition to input variables—branching consistently on the best $p\%$ of unconstrained non-input variables according to criteria that are based on different aspects of the underlying circuit structure. Input variables are always included for assuring that Minisat remains complete under the restrictions; that is, we will *relax the input-restriction*.

We first investigate the following hypothesis.

Hypothesis 5 *The more relaxed the branching restriction is, the better the restriction works with the solver.*

The first relaxation we consider is the *random restriction*:

Random restriction (denoted by $\mathsf{rnd}(p)$): In addition to input variables, branching is allowed on $p\%$ of randomly chosen unconstrained non-input variables.

Intuitively, this results in allowing branching evenly across the underlying circuit structure. The random restriction will also serve as a reference point for the other structural restrictions we consider.

We ran BCMinisat with the random restriction with the percentage values $p = 10, 20, 40, 60, 80$. The results as the cumulative number of solved instances, along with input-restricted and unrestricted branching Minisat, are shown in Figure 9. We observe that, in-line with Hypothesis 5, allowing branching on non-input variables in addition to inputs, the robustness of the branching-restricted Minisat increases gradually.

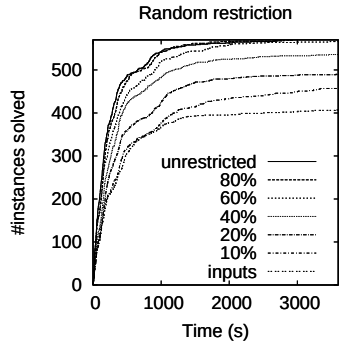


Figure 9: Cumulative number of solved instances for the random branching restriction

We have now seen that the solver benefits from relaxing the input-restriction. However, the random restriction does not take into account structural properties of the selected variables. Following the intuition behind heuristics found in implementations of DPLL without clause learning—based on literal counting [112] for example—we now turn our attention to the following question. In the context of relaxing the input-restriction, how much do structural properties of the variables on which the solver is additionally allowed to branch affect the relative performance of the solver? Our hypothesis is the following.

Hypothesis 6 *Structural properties, based on which branching is restricted, play an important role in the efficiency of the solver.*

We consider various structure-based relaxed branching restrictions in **P2**. Here we discuss two of them: fanout-based restriction $\mathsf{fan}(p)$ and distance-based $\mathsf{maxmin\text{-}dist}(p)$. For the others ($\mathsf{minmax\text{-}dist}(p)$, flow-based restriction $\mathsf{flow}(p)$, and degree-based restriction $\mathsf{deg}(p)$) the reader is referred to **P2**.

For a gate g in a constrained circuit C^α the *fanout* $\mathsf{fanout}(g)$ is the number of gates whose child g or $g' := \mathrm{NOT}(g)$ is. Additionally, let $\Delta^{\min}_{\mathsf{inputs}}(g)$ denote

the length of the shortest path under the child relation of $\mathcal{C}^\alpha$ from g to any input gate. Here NOTs do not contribute to the length of the paths, since they are not translated. Similarly, $\Delta^{\min}_{\mathsf{outputs}}(g)$ stands for the length of the shortest path under the parent relation of $\mathcal{C}^\alpha$ from g to any output gate.

Fanout-based restriction $\mathsf{fan}(p)$**:** Here gates are ranked according to the values $\mathsf{fanout}(g)$, with the criterion that gates with large values are preferred. This is a generalization of the idea of restricting branching to gates g with $\mathsf{fanout}(g) > 1$ as suggested in the context of SAT-based ATPG [219].

Distance-based restrictions: We also consider restricting branching based on the distances of gates from inputs and outputs. In $\mathsf{maxmin\text{-}dist}(p)$ gates are ranked according to the values

$$\min\{\Delta^{\min}_{\mathsf{inputs}}(g), \Delta^{\min}_{\mathsf{outputs}}(g)\},$$

with the criterion that gates with *large* values are preferred. Here the idea is to concentrate branching on variables that are far from both input and output variables.

In selecting the $p\%$ of variables according to a particular criterion, ties are broken randomly so that exactly $p\%$ of all gates are selected.

We ran BCMinisat with all the considered restrictions and values $p = 10, 20, 40, 60, 80$. The results reveal differences between the effectiveness of different restrictions, as exemplified in Figure 10. It is interesting to see that for the fanout and degree-based restrictions only 20% additional branching variables are enough for the restrictions to reach a level of robustness very close to unrestricted branching Minisat. For the flow-based restriction, this holds from 40% on. The distance-based restrictions result in very poor performance, even compared to the random restriction. In accordance with Hypothesis 6, the choice of the structural criterion does make a difference.

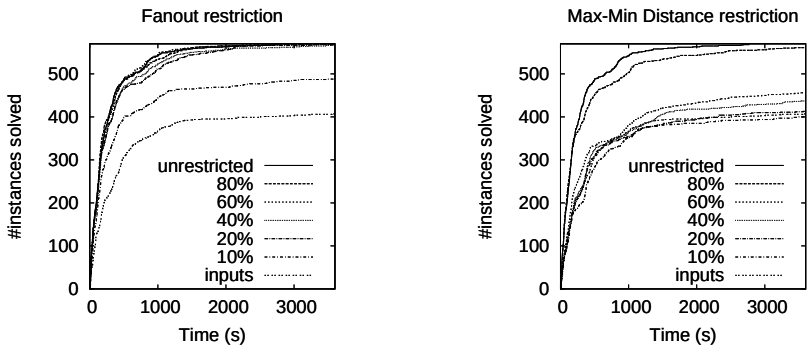


Figure 10: Cumulative number of solved instances for the structural branching restrictions

We look for possible explanations for the fact that the structural property based on which branching is restricted affects the efficiency of the solver. We compare the fanout restriction (very close to the original unrestricted solver in performance), the max-min distance restriction (the worst behaving restriction), and also take the random restriction as a reference. The relative number of branchable variables occurring in the conflict clauses when using these branching restriction criteria are shown in Figures 11 for $p = 20$. We observe a visible difference in the relative number of branchable variables occurring in the conflict clauses. Compared to the other two restrictions, with the fanout restriction the conflict learning mechanism of Minisat produces conflict clauses consisting of a high number of variables on which the solver is allowed to branch.

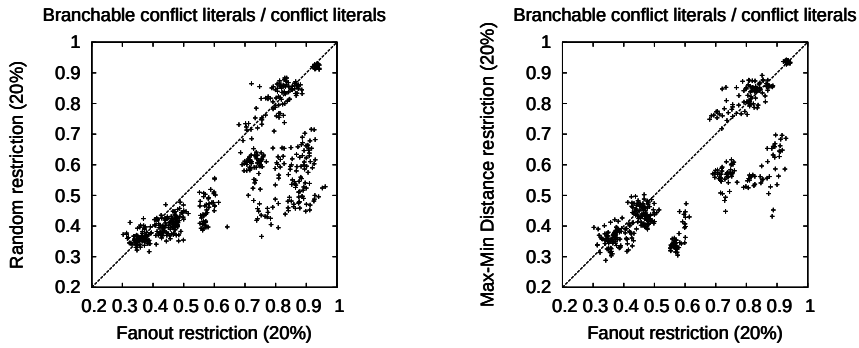


Figure 11: Comparison of fanout, max-min distance, and random restrictions for $p = 20$: relative number of branchable variables occurring in the conflict clauses

Summary

Concerning Hypotheses 1–4, we make the following conclusions based on the experiments on input-restricted branching.

- Although the set of input variables provides a relatively small strong backdoor set, the clause learning SAT solver Minisat, using the VSIDS heuristic, does not benefit from the restriction as such. The performance degrades especially on unsatisfiable instances.
- The effectiveness of clause learning degrades. While the input-restricted branching solver ventures into more conflicts per decision, conflict clauses become longer and more time is used on conflict analysis.
- The solver runs into more conflicts and propagates more per decision. However, this does not help in making the search more efficient, since at the same time the conflict clauses become longer and thus less effective.
- The solver is often forced to branch on variables that are unimportant with respect to heuristic scores of VSIDS.

The main conclusions on relaxed branching restrictions are the following.

- Compared to strict branching restrictions, such as the input-restriction, more relaxed branching restrictions allow the solver to better apply its clause learning and branching heuristics for making search more efficient.
- The choice of the structural criterion based on which branching is restricted plays an important role in the efficiency of the solver; some structural criteria, such as fanout-based, seem to allow rather strict restrictions without loss in efficiency, while other criteria can perform even worse than randomly restricting branching.

We conjecture that the number of variables on which the solver is allowed to branch *in the conflict clauses generated during search* is a determining factor for the efficiency of the branching-restricted solver. Thus, we suggest that when restricting branching in a clause learning solver, it is important to ensure that the conflict clauses generated during search contain a high number of variables on which the solver is allowed to branch. A step into this direction is taken in a recent work [173] which studies this possibility in the special case of At-Most-One cardinality constraints.

4.3 P3 and P4: Proof Complexity Theoretic Limitations of Structure-Based Branching

Considering restricting branching in DPLL, a natural question to ask is whether the power of the underlying inference systems of DPLL-based solvers is affected by the restriction. By forcing input-restricted branching on DPLL without clause learning, this question is answered in [125]: input-restricted branching DPLL cannot polynomially simulate DPLL. The question for DPLL with clause learning is left open. Complementing the experimental results of **P2** on the effects of static branching restrictions on clause learning SAT solving, publication **P3** addresses the proof complexity theoretic power of DPLL-based clause learning SAT solvers with input-restricted branching. For the case of applying dynamic branching restrictions, the proof complexity theoretic effect on DPLL without clause learning studied in [125] considered selected dynamic restrictions. Extending this study, publication **P4** addresses the effect of dynamic "top-down" variants of branching on the proof complexity theoretic power of DPLL-based solvers, both with and without clause learning.

Input-Restricted Branching CL

In the following, we denote input-restricted branching DPLL, CL, and CL– by $\mathsf{DPLL}_{\mathsf{inputs}}$, $\mathsf{CL}_{\mathsf{inputs}}$, and $\mathsf{CL}\text{–}_{\mathsf{inputs}}$, respectively.

In **P3** we show that $\mathsf{CL}_{\mathsf{inputs}}$ and the basic DPLL without clause learning are polynomially incomparable. Hence, $\mathsf{CL}_{\mathsf{inputs}}$ cannot simulate CL. This implies that all implementations of clause learning DPLL, even with optimal heuristics, have the potential of suffering a notable efficiency decrease if branching is restricted to input variables.

The main results of **P3** are summarized in Figure 12. In the figure, an arrow without a slash from system S to S' means that S can polynomially simulate S', and with a slash that S cannot polynomially simulate S'. Those arrow labelled with a reference are known results, and the arrows with the symbol $\star$ are due to trivial subsumption. Disregarding simulation resulting from transitivity, the missing arrows represent open questions.

In more detail, the results in **P3** state that even with unlimited restarts and the ability to create conflicts at will, $\mathsf{CL}_{\mathsf{inputs}}$ cannot simulate the basic DPLL (which does not apply clause learning). This is surprising, since the unrestricted version of this variant of CL can efficiently simulate general resolution [33], being thus very powerful compared to DPLL. On the other hand, DPLL cannot simulate $\mathsf{CL}_{\mathsf{inputs}}$ which has no restarts.

The reason for why DPLL cannot polynomially simulate $\mathsf{CL}_{\mathsf{inputs}}$ follows easily by using any infinite family $\{F_n\}$ of CNF formulas witnessing the fact that DPLL cannot polynomially simulate CL [33]. In more detail, define the family of Boolean circuits $\{\mathsf{circuit}(F) \mid F \in \{F_n\}\}$. For such a family of circuits, $\mathsf{CL}_{\mathsf{inputs}}$ and CL can branch effectively on the same gates, namely those corresponding to the variables occurring in each F_n.

A more intricate construction is needed for achieving a witness for the main result that $\mathsf{CL}\text{–}_{\mathsf{inputs}}$ cannot polynomially simulate DPLL. One key concept in applied in the proof construction is *redundant gates in constrained Boolean circuits*.

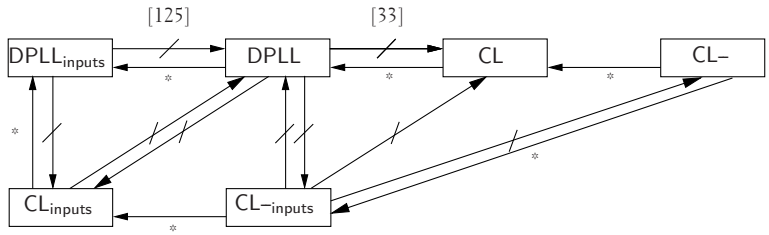


Figure 12: A refined relative efficiency hierarchy for original input-restricted branching variations of DPLL and CL.

Definition 4.1 *A gate in a constrained Boolean circuit $\mathcal{C}^\alpha$ is* redundant *if it is unconstrained and not a descendant of any constrained gate.*

Considering redundant gates, we prove the following lemma in **P3**.

Lemma 4.1 *Let $\mathcal{C}^\alpha$ be an arbitrary constrained Boolean circuit. Considering* CL–$_{\text{inputs}}$ *on input* cnf$(\mathcal{C}^\alpha)$*, redundant gates do not occur in any conflict graph at any stage of* CL–$_{\text{inputs}}$*. This holds whether or not restarts are allowed.*

Intuitively, this is because redundant gates can only have a value due to unit propagation "upwards" (from child to parent) on the circuit structure in CL–$_{\text{inputs}}$; as such gates, and their parents, are not constrained by definition, they cannot cause a conflict or be a part of a unit propagation chain responsible for a conflict. As a consequence, redundant gates can never appear in conflict clauses derived by CL–$_{\text{inputs}}$.

For constructing an infinite family of Boolean circuits which serves as a witness for the fact that CL–$_{\text{inputs}}$ cannot polynomially simulate DPLL, we apply known results on the resolution complexity of a well-known propositional encoding PHP_n^{n+1} of the *pigeon-hole principle*. Namely, it is known that there is no polynomial length RES proof of PHP_n^{n+1} [107]. However, Cook [59] gives a way of introducing a polynomial number of clauses which can be interpreted as a redundant circuit structure $\langle \text{EXT}_n, \emptyset \rangle$ (without any constrained gates) to circuit(PHP_n^{n+1}) so that, contrarily to circuit(PHP_n^{n+1}), the extended circuit

$$\text{circuit}(\text{PHP}_n^{n+1}) \cup \langle \text{EXT}_n, \emptyset \rangle$$

yields a polynomial length proof π in RES. While this does not guarantee a polynomial length proof in T-RES, we add additional structure $\text{E}(\pi)$ to the extended circuit EXT_n: we introduce for each clause C_i in the RES proof π a corresponding OR-gate representing C_i. Again, these additional OR gates are redundant. Through this trick, we end up with the final construct,

$$\text{EPHP}_n^{n+1} := \text{circuit}(\text{PHP}_n^{n+1}) \cup \langle \text{EXT}_n, \emptyset \rangle \cup \langle \text{E}(\pi), \emptyset \rangle,$$

as illustrated in Figure 13.

By adding these two redundant circuit structures to circuit(PHP_n^{n+1}), we assure that there is a simple polynomial length DPLL proof of EPHP_n^{n+1} Intuitively this is because $\text{E}(\pi)$ allows DPLL to "verify" the resolution proof of PHP_n^{n+1} extended with EXT_n step-by-step. However, there is no polynomial

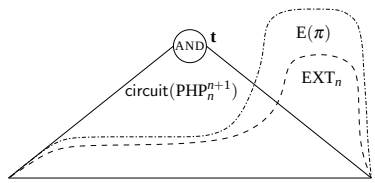


Figure 13: High-level view of EPHP_n^{n+1}.

length proof of EPHP_n^{n+1} in $\mathsf{CL}\text{–}_{\mathsf{inputs}}$. This is because $\mathsf{CL}\text{–}_{\mathsf{inputs}}$ cannot make use of the redundant gates of EXT_n and $\mathrm{E}(\pi)$. This results in the fact that any $\mathsf{CL}\text{–}_{\mathsf{inputs}}$-proof of EPHP_n^{n+1} will effectively contain a proof of PHP_n^{n+1}. Since there is no polynomial length RES proof of PHP_n^{n+1} [107], and RES can polynomially simulate CL– [33], all $\mathsf{CL}\text{–}_{\mathsf{inputs}}$-proof of EPHP_n^{n+1} must be of superpolynomial length.

The Cook's extension (a variant of EXT_n) presented in [59] is motivated by investigations into the power of the *Extended Resolution proof system* defined by Tseitin [233]. Extended Resolution is the result of adding an *extension rule* to RES, which allows for iteratively adding *definitions* of the form $x \leftrightarrow l_1 \wedge l_2$ (or, as a set of clauses, $\{\{x, \neg l_1, \neg l_2\}, \{\neg x, l_1\}, \{\neg x, l_2\}\}$) to the CNF formula, where x is a new variable and l_1, l_2 are literals in the current formula. This is equivalent to adding a redundant binary AND gate of the literals l_1, l_2 to a constrained Boolean circuit. Notably, it is known that Extended Resolution is among the most powerful proof systems, and can simulate, for example, *Frege systems* (see [144] for more details).

Considering the EPHP_n^{n+1} construction, notice that for our purposes, what is most important is that a short RES-proof π exists, not really the actual details of π. For understanding the general idea behind the explicit construction of EPHP_n^{n+1}, it is informative to notice that, instead of the pigeon-hole problem PHP_n^{n+1}, Cook's extension EXT_n to it, and the resolution proof π of their combination, one could use any CNF formula F that (i) does not have a polynomial-length resolution proof but (ii) has a polynomial-length extended resolution proof to prove a result similar to Lemma 3 in **P3**. That is, for such formula F, DPLL has a polynomial length proof of

$$\mathsf{circuit}(F) \cup \langle \mathrm{EXT}_F, \emptyset \rangle \cup \langle \mathrm{E}(\pi_F), \emptyset \rangle$$

while $\mathsf{CL}\text{–}_{\mathsf{inputs}}$ does not, where EXT_F is the polynomial size extension of F and π_F is a polynomial-length resolution proof of

$$\mathsf{cnf}(\mathsf{circuit}(F) \cup \langle \mathrm{EXT}_F, \emptyset \rangle).$$

Top-Down Branching Variants of DPLL and CL

Publication **P4** presents a study of the proof complexity theoretic effects of applying so called "top-down" dynamic branching restrictions in DPLL and CL. The idea is to apply branching in a top-down fashion as follows: starting from the constraints imposed on the output gates of the circuit, search for justification for the currently assigned values [150, 166]. Additionally, in conjunction with top-down branching, a modification to the actual *style of branching* in DPLL-based algorithms, aiming at eagerly justifying the currently unjustified

gates [149], is also considered. We refer to this as *ATPG-style branching* in **P4**. Although somewhat misleading, this naming comes from the fact that in [149], automatic test pattern generation (ATPG) is mentioned as one of the main application areas of their approach.

As the main result in **P4** we present a relative efficiency hierarchy for variations of circuit-level DPLL (with and without clause learning) resulting from combinations of branching heuristics and branching styles. The variations are

(i) DPLL-style top-down restricted (as considered in [125] for basic DPLL without clause learning),

(ii) DPLL-style justification restricted [150, 166], and

(iii) ATPG-style justification restricted [149] branching DPLL.

As in solver implementations, we assume that, in the above-mentioned variations of DPLL and CL, unit propagation is applied whenever applicable.

We characterize the variants of top-down branching DPLL and CL through two dynamic branching restrictions.

Top-down restriction: Branching is allowed on gate g if g has a currently assigned parent. These variants of DPLL and CL are denoted by $\mathsf{DPLL}_{\mathsf{td}}$ and $\mathsf{CL}_{\mathsf{td}}$.

Justification-based restriction: Branching is allowed on gate g if g has a currently assigned and unjustified parent. These variants of DPLL and CL are $\mathsf{DPLL}_{\mathsf{jf}}$ and $\mathsf{CL}_{\mathsf{jf}}$.

As an example of the difference between the top-down and justification based restrictions, consider an OR-gate $g := \mathrm{OR}(g_1, g_2, g_3)$. In the case g is currently assigned to **t**, with the top-down restriction we are allowed to branch on the unassigned children of g. In contrast, with the justification-based restriction, we are allowed to branch on the unassigned children of g only if none of the children are currently assigned to **t**.

The underlying $\mathsf{DPLL}_{\mathsf{jf}}^{\mathsf{atpg}}$ system using ATPG-style branching is a variation of the justification-based restricted branching $\mathsf{DPLL}_{\mathsf{jf}}$. The difference between original DPLL-style branching and ATPG-style branching is illustrated in Figure 14 with an OR-gate $g := \mathrm{OR}(g_1, g_2, g_3)$. Where original DPLL-style branching is based on branching on a variable (Figure 14 left), in ATPG-style branching (Figure 14 right) each branch will have a unique minimal justification for the currently assigned value of the parent (g is **t** in the example).

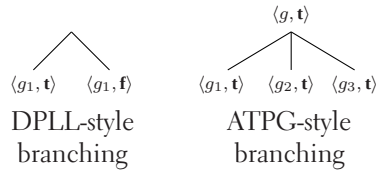


Figure 14: Styles of branching; OR-gate $g := \mathrm{OR}(g_1, g_2, g_3)$

Notice that ATPG-style branching is similar to the decomposition rules applied in Smullyan's analytic tableaux [222], and, depending on context, could also be called *syntactic branching* [94]. On the other hand, DPLL-style branching could be called *binary branching* or *semantic branching* [94]. We also note that the reason for not considering $\mathsf{CL}^{\mathsf{atpg}}_{\mathsf{jf}}$ is that in contrast to DPLL, when branching in CL by assigning a gate to a specific value, the opposite value of the gate is not necessarily forced systematically through backtracking.

The results of **P4** are summarized in Figure 15. For example, for DPLL without clause learning, we establish a strict hierarchy, with the ATPG-style branching, justification restricted DPLL variant ($\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$) being the weakest system. Perhaps the most surprising result obtained is that clause learning DPLL with justification restricted decision heuristics ($\mathsf{CL}_{\mathsf{jf}}$) cannot even simulate the top-down restricted variant *without clause learning* ($\mathsf{DPLL}_{\mathsf{td}}$). Thus, although the idea of eagerly and locally justifying the values of currently unjustified constraints is an intuitively appealing one, it can lead to dramatic losses in the best-case efficiency of a structure-aware SAT solver even when the powerful search space pruning technique of clause learning is applied. It is also worth noticing that, considering the variants of CL, the results hold also in the case the systems are all allowed restarts.

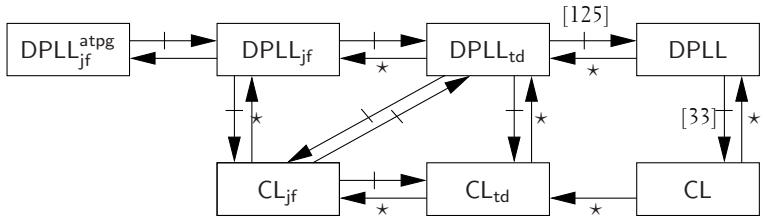


Figure 15: Refined relative efficiency hierarchy for original top-down branching variations of DPLL and CL.

For some intuition behind the proofs, in order to separate $\mathsf{DPLL}_{\mathsf{jf}}$ and $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$, for example, we use a known result on the efficiency of *clausal tableaux* (CT), a tableau proof system for CNF formulas. Namely, it is known that CT cannot polynomially simulate T-RES [18]. As is shown in **P4**, $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ and CT are in fact polynomially equivalent on CNF formulas, and, on the other hand, $\mathsf{DPLL}_{\mathsf{jf}}$ is polynomially equivalent to DPLL and hence to T-RES on CNF formulas, it follows that $\mathsf{DPLL}^{\mathsf{atpg}}_{\mathsf{jf}}$ cannot polynomially simulate $\mathsf{DPLL}_{\mathsf{jf}}$ follows.

The circuit construction applied in showing that $\mathsf{DPLL}_{\mathsf{jf}}$ cannot polynomially simulate $\mathsf{DPLL}_{\mathsf{td}}$ is based on the EPHP^{n+1}_{n} family of circuits used in **P3**. The $\mathsf{circuit}(\mathrm{PHP}^{n+1}_{n})$ and EXT_n remain intact. However, in order to assure that $\mathsf{DPLL}_{\mathsf{td}}$ can (in a top-down fashion) "verify" the polynomial RES proof as in the case of DPLL for EPHP^{n+1}_{n} in **P3**, we modify the $\mathrm{E}(\pi)$ part of EPHP^{n+1}_{n}. The modified version $\mathrm{P}(\pi)$ of $\mathrm{E}(\pi)$ is illustrated in Figure 16. The idea here is to introduce gates of the form $h_i := \mathrm{AND}(g_{C_i}, h_{i+1})$ in addition to the gates g_{C_i} in $\mathrm{E}(\pi)$ representing the ith clause C_i in π.

While we refer to **P4** for the particulars, the intuitive idea is that $\mathsf{DPLL}_{\mathsf{td}}$

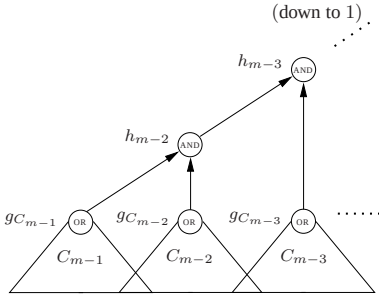


Figure 16: The $\mathrm{P}(\pi)$ circuit construct.

can (in a top-down fashion) "verify" the polynomial RES proof for EPHP_n^{n+1} embedded in the $\mathrm{P}(\pi)$ part of the circuit

$$\mathsf{circuit}(\mathrm{PHP}_n^{n+1}) \cup \langle \mathrm{EXT}_n, \emptyset \rangle \cup \langle \mathrm{P}(\pi), \emptyset \rangle.$$

Since our aim is to show that $\mathsf{DPLL}_{\mathsf{jf}}$ cannot polynomially simulate $\mathsf{DPLL}_{\mathsf{td}}$, we additionally introduce a simple constrained circuit structure on top of the output gates of $\mathrm{P}(\pi)$ and $\mathsf{circuit}(\mathrm{PHP}_n^{n+1})$. The final resulting circuits construct, PPHP_n^{n+1}, is illustrated in Figure 17. Now, $\mathsf{DPLL}_{\mathsf{td}}$ can (in a top-down fashion) "verify" the polynomial RES proof for EPHP_n^{n+1} embedded in the $\mathrm{P}(\pi)$ part of PPHP_n^{n+1}. However, $\mathsf{DPLL}_{\mathsf{jf}}$ can never branch on the gates in $\mathrm{P}(\pi)$ which would be necessary for achieving a polynomial-length proof for PPHP_n^{n+1}. Again, the intuition here is that any $\mathsf{DPLL}_{\mathsf{jf}}$-proof of PPHP_n^{n+1} must contain a (necessarily superpolynomial-length) proof of PHP_n^{n+1}. In fact, even $\mathsf{CL}_{\mathsf{jf}}$ has not polynomial-length proofs for the circuit construction based on EPHP_n^{n+1}, which gives us the result that even $\mathsf{CL}_{\mathsf{jf}}$ cannot polynomially simulate $\mathsf{DPLL}_{\mathsf{td}}$.

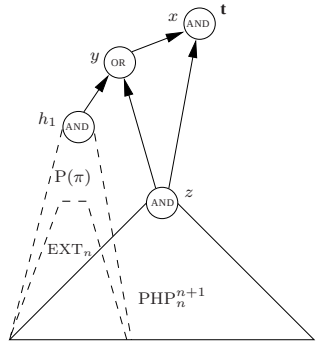


Figure 17: High-level view of PPHP_n^{n+1}.

For further work, an interesting question left open in **P4** (recall Figure 15), is whether $\mathsf{CL}_{\mathsf{td}}$ can simulate CL (or DPLL).

4.4 P5: Extended ASP Tableaux and Redundancy in ASP

Although ASP solvers for normal logic programs have been available for many years [220, 17, 153, 82], the deduction rules applied in such solvers have only recently been formally defined as a proof system, which we will here refer to as ASP Tableaux or ASP-T [84]. ASP-T is a sound and complete tableau proof system for normal logic programs, that is, there is a complete non-contradictory ASP tableau for a NLP Π if and only if Π is satisfiable [84]. As argumented in [84], current ASP solver implementations are tightly related to ASP-T, with the intuition that the branching (cut) rule is made deterministic with decision heuristics, while the deduction rules describe the propagation mechanism in ASP solvers. This is very similar to DPLL-based SAT solvers.

As typical for tableau-based proof systems, an ASP tableau for a NLP Π is a binary tree of the following structure. The *root* of the tableau consists of the rules Π and the *entry* $\mathbf{F}\bot$ for capturing that $\bot$ is always false. The non-root nodes of the tableau are single *entries* of the form $\mathbf{T}a$ or $\mathbf{F}a$, where $a \in \mathsf{atoms}(\Pi) \cup \mathsf{body}(\Pi)$. As typical for tableau methods, entries are generated by *extending* a *branch* (a path from the root to a leaf node) by applying one of the rules in Figure 1 in **P5**; if the prerequisites of a rule hold in a branch, the branch can be extended with the entries specified by the rule. However, we will not discuss the various deduction rules in ASP-T here in detail. While we refer the reader to **P5** for details, the following example aims at giving an intuitive idea of ASP-T.

Example 4.1 *An* ASP-T *proof (a closed* ASP-T *tableau) for the program*

$$\Pi = \{a \leftarrow b, {\sim}a.\ b \leftarrow c.\ c \leftarrow {\sim}b\}$$

is shown in Figure 18, with the rule applied for deducing each entry given in parentheses. For example, the entry $\mathbf{F}a$ *has been deduced from* $a \leftarrow b, {\sim}a$ *in* Π *and the entry* $\mathbf{T}\{b, {\sim}a\}$ *in the left branch by applying the rule (g) Backward True Body. On the other hand,* $\mathbf{T}\{b, {\sim}a\}$ *has been deduced from* $a \leftarrow b, {\sim}a$ *in* Π *and the entry* $\mathbf{T}a$ *in the left branch by applying the rule (i§), that is, rule (i) by the fact that the condition § "Backward True Atom" is fulfilled (in* Π*, the only body with atom* a *in the head is* $\{b, {\sim}a\}$*). The tableau in Figure 18 has two closed (contradictory) branches:*

$$(\Pi \cup \{\mathbf{F}\bot\}, \mathbf{T}a, \mathbf{T}\{b, {\sim}a\}, \mathbf{F}a) \text{ and}$$

$$(\Pi \cup \{\mathbf{F}\bot\}, \mathbf{F}a, \mathbf{F}\{b, {\sim}a\}, \mathbf{F}b, \mathbf{T}\{{\sim}b\}, \mathbf{T}c, \mathbf{T}\{c\}, \mathbf{T}b).$$

As discussed in detail in **P5**, ASP-T and T-RES are in fact polynomially equivalent under the translations comp and nlp. Although the similarity of unit propagation in DPLL and propagation in ASP solvers is discussed in [90, 83], we stress the direct connection between ASP-T and T-RES. In detail, T-RES and ASP-T are equivalent in the sense that (i) given an arbitrary tight NLP Π, the minimum-length proofs for $\mathsf{comp}(\Pi)$ in T-RES are polynomially bounded with respect to the minimum-length proofs for Π in ASP-T, and

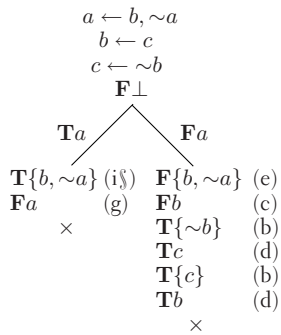


Figure 18: An ASP-T proof for $\Pi = \{a \leftarrow b, \sim a.\ b \leftarrow c.\ c \leftarrow \sim b\}$.

(ii) given an arbitrary CNF formula F, the minimum-length proofs for $\mathsf{nlp}(F)$ in ASP-T are polynomially bounded with respect to the minimum-length proofs for F in T-RES.

Motivated by the very powerful extended resolution proof system [233] for CNF SAT, in publication **P5** we introduce an *extension rule* to ASP-T, which results in *Extended ASP Tableaux* (E-ASP-T), an extended tableau proof system for ASP.

The idea of the *extension rule* for ASP-T is that one can define names for conjunctions of default literals.

Definition 4.2 *Given a normal logic program* Π *and two literals* $l_1, l_2 \in \mathsf{dlits}(\Pi)$*, the (elementary)* extension rule *for* ASP-T *adds the rule* $p \leftarrow l_1, l_2$ *to* Π*, where* $p \notin \mathsf{atoms}(\Pi) \cup \{\bot\}$.

It is essential that p is a new atom for preserving satisfiability. After an application of the extension rule one considers program $\Pi' = \Pi \cup \{p \leftarrow l_1, l_2\}$ instead of the original program Π. Notice that $\mathsf{atoms}(\Pi') = \mathsf{atoms}(\Pi) \cup \{p\}$. Thus when the extension rule is applied several times, the atoms introduced in previous applications of the rule can be used in defining further new atoms (see Example 4.2 below).

When convenient, one can apply a generalization of the elementary extension. By allowing one to introduce multiple bodies for p, the *general extension rule* adds a set of rules

$$\bigcup_i \{p \leftarrow l_{i,1}, \ldots, l_{i,k_i} \mid p \notin \mathsf{atoms}(\Pi) \cup \{\bot\} \text{ and } l_{j,k} \in \mathsf{dlits}(\Pi)\}$$

into Π. Notice that equivalent constructs can be introduced with the elementary rule. For example, bodies with more than two literals can be decomposed with balanced parentheses using additional new atoms.

Example 4.2 *Consider a normal logic program* Π *such that* $\mathsf{atoms}(\Pi) = \{a, b\}$*. We apply the general extension rule and add a definition for the disjunction of atoms* a *and* b*, resulting in the program*

$$\Pi \cup \{c \leftarrow a.\ c \leftarrow b\ \}.$$

An equivalent construct can be introduced by applying the elementary extension rule twice: first add the rule $d \leftarrow \sim a, \sim b$, and then the rule $c \leftarrow \sim d, \sim d$.

An E-ASP-T proof of program Π is an ASP-T proof T of $\Pi \cup E$, where E is a set of *extending (program) rules* generated with the extension rule in E-ASP-T. The length of an E-ASP-T proof is the length of T plus the number of program rules in E.

A key point is that applications of the extension rule do not affect the existence of stable models. In other words, E-ASP-T is sound and complete, as detailed in **P5**.

Extended ASP Tableaux and Extended Resolution
We relate E-ASP-T with E-RES in **P5** by showing in detail that these two proof systems are polynomially equivalent under the translations comp and nlp.

Theorem 4.2 E-RES *and* E-ASP-T *are polynomially equivalent proof systems in the sense that*

(i) considering tight normal logic programs, E-RES *under the translation* comp *polynomially simulates* E-ASP-T*, and*

(ii) considering CNF formulas, E-ASP-T *under the translation* nlp *polynomially simulates* E-RES*.*

The proof details are rather similar to the key points used in **P3** when arguing why DPLL has short proofs for the EPHP_n^{n+1} family of CNF formulas. In other words, one can think of the proof of part (ii) of Theorem 4.2 as an interpretation of the main proof constructs applied in **P3** in the context of ASP.

The Extension Rule and Well-Founded Deduction
An interesting question regarding the possible gains of applying the extension rule for ASP-T with the ASP tableau rules is whether the additional extension rule allows one to simulate well-founded deduction (See ASP-T rules (h†),(h‡),(i†), and (i‡) in **P5**) with the other deduction rules (See ASP-T rules (b)–(g),(h§),(i§) in **P5**; for tight NLPs, these rules are equivalent to unit propagation on the clausal completion of a program). We show that this is not the case; the extension rule does not allow us to simulate reasoning related to unfounded sets and loop formulas. In more detail, by removing rules (h†),(h‡),(i†), and (i‡) from E-ASP-T, the resulting tableau method becomes incomplete for NLPs.

Theorem 4.3 *The tableau rules (a)–(g), (h§), and (i§) in addition to the extension rule do not result in a complete proof system for normal logic programs under stable model semantics.*

Experiments
In addition to the theoretical results, in **P5** we experimentally evaluate how well current state-of-the-art ASP solvers can make use of the additional structure introduced to programs using the extension rule. For the experiments,

we ran the solvers smodels [220] (a widely used lookahead solver), clasp [82] (with many techniques—including conflict learning—adopted from DPLL-based SAT solvers), and cmodels [89] (a SAT-based ASP solver running the clause learning SAT solver zChaff as the back-end).

We compare the number of decisions and running times of each of the solvers on the NLP representation of PHP_n^{n+1} and EPHP_n^{n+1}. Additionally, a family CPHP_n^{n+1} of NLPs is considered, for which $\mathrm{PHP}_n^{n+1} \subset \mathrm{CPHP}_n^{n+1} \subset \mathrm{EPHP}_n^{n+1}$ holds; however, we refer the reader to Section 6 of **P5** for details on CPHP_n^{n+1}.

While the number of decisions for the conflict-learning solvers clasp and cmodels is somewhat reduced by the extensions, the solvers do not seem to be able to reproduce the polynomial size proofs that exist for EPHP_n^{n+1}, and we do not observe a dramatic change in the running times compared to PHP_n^{n+1}. With a timeout of 2 hours, smodels gives no answer for $n = 12$ on PHP_n^{n+1}. However, on EPHP_n^{n+1}, smodels returns without any branching, which is due to the fact that smodels, using lookahead, can find the short proof which "verifies" the polynomial resolution proof encoded in EPHP_n^{n+1}. For the detailed results, we refer the reader to Table 1 of **P5**.

In the second experiment, we study the effect of having a modest number of redundant rules on the behavior of ASP solvers. For this we apply the procedure AddRandomRedundancy(Π, n, p) shown in Algorithm 2.

Algorithm 2 AddRandomRedundancy(Π, n, p)

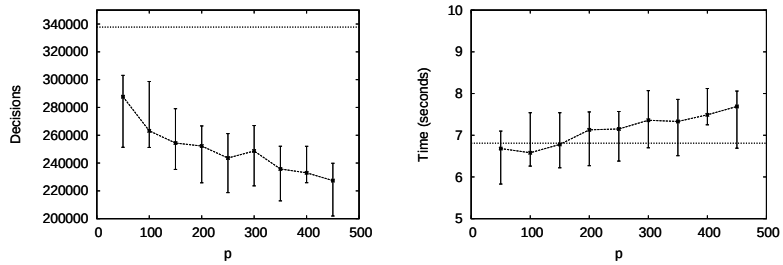
1. **For** $i = 1$ **to** $\lfloor \frac{p}{100} n \rfloor$:
   - 1a. Randomly select $l_1, l_2 \in \mathsf{dlits}(\Pi)$ such that $l_1 \neq l_2$.
   - 1b. $\Pi := \Pi \cup \{r_i \leftarrow l_1, l_2\}$, where $r_i \notin \mathsf{atoms}(\Pi) \cup \{\bot\}$.
2. **Return** Π

Given a program Π, the procedure iteratively adds rules of the form $r_i \leftarrow l_1, l_2$ to Π, where l_1, l_2 are random default literals currently in the program and r_i is a new atom. The number of introduced rules is $p\%$ of the integer n.

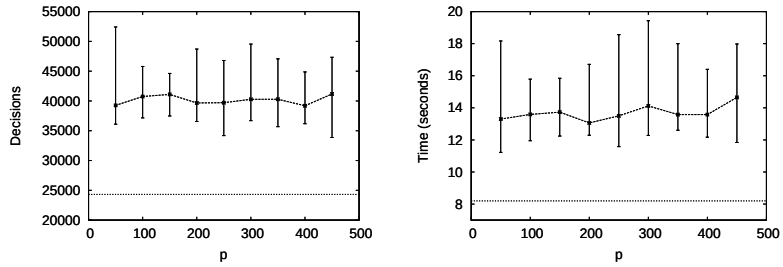
The median, minimum, and maximum number of decisions and running times for the solvers on AddRandomRedundancy$(\mathrm{PHP}_n^{n+1}, n, p)$ are shown in Figure 19 for the percentages $p = 50, 100 \ldots, 450$ over 15 trials for each value of p. The mean number of decisions (left) and running times (right) on the original PHP_n^{n+1} are presented by the horizontal lines. Notice that the number of added atoms and rules is linear to n, which is negligible to the number of atoms (in the order of n^2) and rules (n^3) in PHP_n^{n+1}. For similar running times, the number of holes n is 10 for clasp and smodels and 11 for cmodels. The results are interesting: each of the solvers seems to react individually to the added redundancy. For cmodels (b), only a few added redundant rules are enough to worsen its behavior. For smodels (c), the number of decisions decreases linearly with the number of added rules. However, the running times grow fast at the same time, most likely due to smodels' lookahead.

The most interesting effect is seen for clasp; clasp benefits from the added

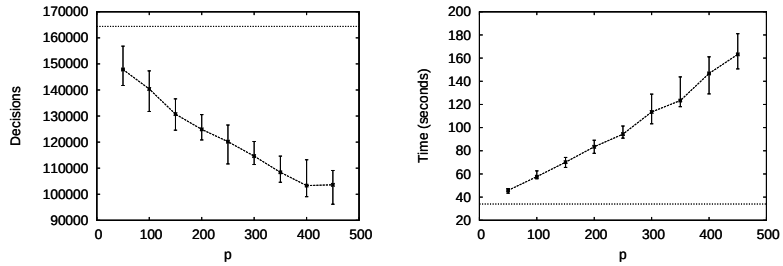
rules with respect to the number of decisions, while the running times stay similar on the average, contrarily to the other solvers. In addition to this robustness against redundancy, we believe that this shows promise for further exploiting redundancy added in a controlled way during search; the added rules give new possibilities to branch on definitions which were not available in the original program. However, for benefiting from redundancy with respect to running times, optimized lightweight propagation mechanisms are essential.



(a) clasp decisions (left), time in seconds (right)



(b) cmodels decisions (left), time in seconds (right)



(c) smodels decisions (left), time in seconds (right)

Figure 19: Effects of adding randomly generated redundant rules to PHP_n^{n+1}

4.5 P6 and P7: Structure-Based Techniques for Stochastic Local Search

In **P6** and **P7** a novel non-clausal SLS method for structured real-world SAT instances is developed. The aim is to bring structure-exploiting techniques into local search for SAT in order to lift the performance of local search SAT solving especially on structured real-world problem domains. We employ Boolean circuits as the representation of general propositional formulas. Motivated by *justification frontier* heuristics (see [150] for example) applied in complete circuit-level SAT solvers in electronic design automation, our search technique looks for a *justification* for the Boolean circuit instead of focusing on finding a satisfying truth assignment. The idea is to be able to drive local search more top-down in the overall structure of the circuit rather than in a bottom-up mode as is done in local search techniques focusing on input variables. This is done by guiding the search using justification frontiers that enable exploiting *observability don't cares* (see [204] for example), drive the search to relevant parts of the circuit, and offer early stopping criteria which allow to end the search when the circuit is *de facto satisfiable* even if no concrete satisfying truth assignment has been found.

BC SLS

The BC SLS framework is described as Algorithm 3. Given a constrained Boolean circuit $\mathcal{C}^\alpha$, the algorithm performs structure-based local search over the assignment space of *all* the gates in $\mathcal{C}$ (inner loop on lines 3–16). As typical, the *noise parameter* $p \in [0,1]$ controls the probability of making non-greedy moves (with $p = 0$ only greedy moves are made). We also introduce an additional parameter $q \in [0,1]$ which leads to a *probabilistically approximately complete* (PAC) generalization of BC SLS. We note that this general BC SLS is the version presented in **P7**, having an additional random choice in the non-greedy move controlled by q. The version described in **P6** results from fixing $q = 0$.

For each of the $\textsc{MaxTries}(\mathcal{C}^\alpha)$ runs, $\textsc{MaxMoves}(\mathcal{C}^\alpha)$ moves are made. As the *stopping criterion* we use the condition that the justification frontier $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty. As discussed in Section 2.1, if $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ is empty, then $\mathcal{C}^\alpha$ is satisfiable and a satisfying truth assignment can be computed in linear-time from τ. Notice that typically this stopping criterion is reached before all gates are locally justified in the current configuration τ.

Given the current configuration τ, we focus moves on locally justifying gates in $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ by randomly picking a gate g from this set. For a gate g and its current value v in τ, the possible *greedy moves* are induced by the justifications for $\langle g, v\rangle$. The idea is to minimize the *size of the interest set*. In other words, the value of the objective function for a move (justification) δ is

$$\mathsf{cost}(\tau, \delta) = |\mathsf{interest}(\mathcal{C}^\alpha, \tau')|,$$

where $\tau' = (\tau \setminus \{\langle g, \neg w\rangle \mid \langle g, w\rangle \in \delta\}) \cup \delta$. In other words, the cost of a move δ is the size of the interest set in the configuration τ' where for the gates mentioned in δ we use the values in δ instead of those in τ. The move is then selected randomly from those justifications δ for $\langle g, v\rangle$ for which $\mathsf{cost}(\tau, \delta)$ is smallest over all justifications for $\langle g, v\rangle$.

Algorithm 3 General BC SLS

Input: constrained Boolean circuit $\mathcal{C}^\alpha$, control parameters $p, q \in [0,1]$ for non-greedy moves
Output: a *de facto* satisfying assignment for $\mathcal{C}^\alpha$ or "*don't know*"
Explanations:
τ: current truth assignment on all gates with $\tau \supseteq \alpha$
δ: next move (a partial assignment)

1: **for** $try := 1$ to MAXTRIES$(\mathcal{C}^\alpha)$ **do**
2: $\quad\tau :=$ pick an assignment over all gates in $\mathcal{C}$ s.t. $\tau \supseteq \alpha$
3: $\quad$**for** $move := 1$ to MAXMOVES$(\mathcal{C}^\alpha)$ **do**
4: $\quad\quad$**if** $\mathsf{jfront}(\mathcal{C}^\alpha, \tau) = \emptyset$ **then return** τ
5: $\quad\quad$Select a random gate $g \in \mathsf{jfront}(\mathcal{C}^\alpha, \tau)$
6: $\quad\quad$**with** probability $(1-p)$ **do** *%greedy move*
7: $\quad\quad\quad\delta :=$ a random justification from those justifications for $\langle g, v\rangle \in \tau$ that minimize $\mathsf{cost}(\tau, \cdot)$
8: $\quad\quad$**otherwise** *%non-greedy move (with probability p)*
9: $\quad\quad\quad$**if** g is constrained in α
10: $\quad\quad\quad\quad\delta :=$ a random justification for $\langle g, v\rangle \in \tau$
11: $\quad\quad\quad$**else**
12: $\quad\quad\quad\quad$**with** probability $(1-q)$ **do**
13: $\quad\quad\quad\quad\quad\delta := \{\langle g, \neg\tau(g)\rangle\}$ *%flip the value of g*
14: $\quad\quad\quad\quad$**otherwise**
15: $\quad\quad\quad\quad\quad\delta :=$ a random justification for $\langle g, v\rangle \in \tau$
16: $\quad\quad\tau := (\tau \setminus \{\langle g, \neg w\rangle \mid \langle g, w\rangle \in \delta\}) \cup \delta$
17: **return** "*don't know*"

For *non-greedy moves* (lines 9–15, executed with probability p), the control parameter q defines the probability of justifying the selected gate g by a randomly chosen justification from the set of all justifications for the value of g (this is a *non-greedy downward move*). With probability $(1-q)$ the non-greedy move consists of inverting the value of *the gate g itself* (a *non-greedy upward move*). The idea in upward moves is to try to escape from possible local minima by more radically changing the justification frontier. In the special case when g is constrained in α, a random downward move is done with probability 1.

Notice that the size of the interest set gives an upper bound on the number of gates that still need to be justified (the descendants of the gates in the frontier). Following this intuition, by applying the objective function of minimizing the size of the interest set, the greedy moves drive the search towards the input gates. Alternatively, one could use the objective of minimizing the size of the justification frontier since moves are concentrated on gates in the frontier and since the search is stopped when the frontier is empty. However, we notice that the size of the interest set is more responsive to quantifying the changes in the configuration than the size of the justification frontier, as exemplified in Figure 20. The size of the justification frontier typically drops rapidly close to zero from its starting value (the y axis is scaled to $[0,1]$ in the figure), and after this remains quite stable until a solution is found. This is very similar to the typical behavior observed for objective functions based on

the number of unsatisfied clauses in CNF-level SLS methods [214]. In contrast, the size of the interest set can vary significantly without visible changes in the size of the justification frontier.

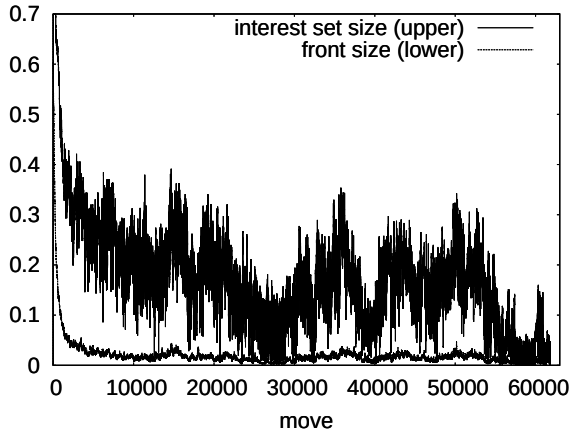


Figure 20: Comparison of dynamics: sizes of interest set and justification frontier

Comparison with CNF-Level SLS Methods

One of the main advantages of the proposed BC SLS method over CNF-level local search methods is that BC SLS can exploit observability don't cares. As an example, consider the circuit in Figure 21, where the gate g_1 is constrained to true and the other **t** and **f** symbols depict the current configuration τ.

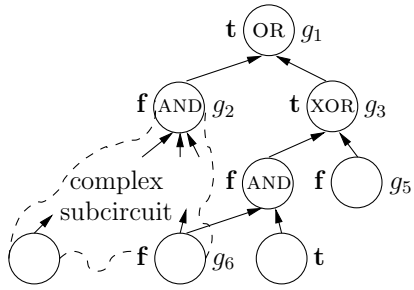


Figure 21: Exploiting don't cares

All the gates, except g_6, in the complex subcircuit rooted at the gate g_2 are don't cares under τ. Therefore BC SLS can ignore the subcircuit and terminate after flipping the input gate g_5 as the justification frontier becomes empty. On the other hand, assume that we translate the circuit into a CNF formula by using the Tseitin translation cnf (recall Section 2.3). If we apply a CNF-level SLS algorithm such as WalkSAT on the CNF formula, observability don't cares are no longer available in the sense that the algorithm must

find a total truth assignment that simultaneously satisfies all the clauses originating from the subcircuit. This can be a very complex task.

We can also analyze how BC SLS behaves on flat CNF input. To do this, we associate a CNF formula $F = C_1 \wedge \ldots \wedge C_k$ with a *constrained CNF circuit* $\mathsf{ccirc}(F) = \langle \mathcal{C}, \alpha \rangle$ as follows. Take an input gate g_x for each variable x occurring in F. Now

$$\begin{aligned} \mathcal{C} \;=\; & \{g_{C_i} := \mathrm{OR}(g_{l_1}, ..., g_{l_m}) \mid C_i = (l_1 \vee \ldots \vee l_m)\} \cup \\ & \{g_{\neg x} := \mathrm{NOT}(g_x) \mid \neg x \in \cup_{i=1}^{k} C_i\} \end{aligned}$$

and the constraints force each "clause gate" g_{C_i} to true, that is, we set $\alpha = \{\langle g_{C_i}, \mathbf{t} \rangle \mid 1 \leq i \leq k\}$. This is illustrated in Figure 22 for $F = (x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3 \vee x_4)$.

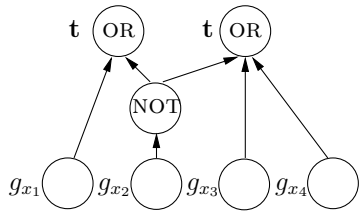


Figure 22: A CNF circuit

When BC SLS is run on a CNF circuit, it can only flip input variables. If input gates were excluded from the set $\mathsf{interest}(\mathcal{C}^\alpha, \tau)$ of interesting gates, then $|\mathsf{interest}(\mathcal{C}^\alpha, \tau)|$ would equal to the number of unjustified clause gates in the configuration τ. Thus the greedy move cost function $\mathsf{cost}(\tau, \cdot)$ would equal to that applied in WalkSAT measuring the number of clauses that are fixed/broken by a flip. Since input gates are included in $\mathsf{interest}(\mathcal{C}^\alpha, \tau)$, the BC SLS cost function also measures, in CNF terms, the number of variables occurring in unsatisfied clauses.

Comparison with Non-Clausal Methods

SLS techniques working directly on non-clausal problems closest to our work include [208, 137, 187]. They are all based on the idea of limiting flipping to input (independent) variables whereas we allow flipping all gates (subformulas) of the problem instance. Moreover, in these approaches the greedy part of the search is driven by a cost function which is substantially different from the justification-based cost function that we employ. Sebastiani [208] generalizes the GSAT heuristic to general propositional formulas and defines the cost function by (implicitly) considering the CNF form *cnf*(ϕ) of the general formula ϕ: the cost for a truth assignment is the number of clauses in *cnf*(ϕ) falsified by the assignment. The approaches of Kautz and Selman [137] and Pham et al. [187] both use a Boolean circuit representation of the problem and employ a cost function which, given a truth assignment for the input gates, counts the number of constrained output gates falsified by the assignment. This cost function provides limited guidance to greedy moves in cases

where there are few constrained output gates or they are far from the input gates. A worst-case scenario occurs when the Boolean circuit given as input has *a single output gate* implying that the cost function can only have the values 0 or 1 for any flip under any configuration. Such a cost function does not offer much direction for the greedy flips towards a satisfying truth assignment. The cost function in BC SLS appears to be less sensitive to the number of output gates or their distance from the input gates. This is because the search is based on the concept of a justification frontier which is able to distribute the requirements implied by the constrained output gates deeper in the circuit.

On Probabilistically Approximate Completeness

We analyze under which conditions BC SLS is PAC (*probabilistically approximately complete*) [113]. A CNF-level SLS SAT method S is PAC if, for any satisfiable CNF SAT instance F and any initial configuration τ, the probability that S *eventually* finds a satisfying truth assignment for F starting from τ is 1 *without using restarts*, i.e., the number of allowed flips is set to infinity and the number of tries to one. A non-PAC SLS method is *essentially incomplete*. Examples of PAC CNF-level SLS methods include GWSAT (with non-zero random walk probability) and UnitWalk, while GSAT, WalkSAT/TABU and Novelty (for arbitrary noise parameter setting) are essentially incomplete [113, 111]. Here we adapt the definition of PAC to the context of BC SLS.

Definition 4.3 *BC SLS is PAC using fixed parameters p, q if, for any satisfiable constrained circuit $\mathcal{C}^\alpha$ and any initial configuration τ, the probability that BC SLS eventually finds a de facto satisfying assignment for $\mathcal{C}^\alpha$ starting from τ is 1 when setting* $\textsc{MaxTries}(\mathcal{C}^\alpha) = 1$ *and* $\textsc{MaxMoves}(\mathcal{C}^\alpha) = \infty$.

It turns out that for a PAC variant of BC SLS, both upward and downward non-greedy moves are needed.

Theorem 4.4 *The variant of BC SLS where non-greedy downward moves are allowed with probability q, where $0 < q < 1$, is PAC for any fixed noise parameter $p > 0$.*

Interestingly, downward non-greedy moves can be restricted to *minimal* justifications without affecting Theorem 4.4. However, if non-greedy moves are only allowed either (i) upwards or (ii) downwards, then BC SLS becomes essentially incomplete.

Theorem 4.5 *The variant of BC SLS where non-greedy moves are done only upwards (that is, when $q = 0$) is essentially incomplete for any fixed noise parameter p.*

Theorem 4.6 *The variant of BC SLS where non-greedy moves are done only downwards (that is, when $q = 1$) is essentially incomplete for any fixed noise parameter p.*

Experimental Results Related to P6

In order to evaluate the ideas behind the BC SLS framework in **P6**, we implemented a prototype of BC SLS on top of the `bc2cnf` Boolean circuit simplifier/CNF translator [131]. In this first prototype, the computation of justification cone is implemented directly by the definition.

As an implementational decision, when making greedy and random moves justifications are selected from the set of subset minimal justifications for the gate value; for a true OR-gate and false AND-gate, the value of a single child is inverted, and for a true OR-gate and false AND-gate the values of all children are inverted.

As structural benchmarks we use a set of Boolean circuits encoding bounded model checking of asynchronous systems for deadlocks [108][3]. Although rather easy for DPLL-based solvers, these benchmarks are challenging for typical SLS methods.

For comparing BC SLS with CNF-level SLS methods, we apply exactly the same Boolean circuit level simplification in bc2cnf to the circuits as in our prototype (including, for example, circuit-level propagation that is equivalent to unit propagation), and then translate the simplified circuit to CNF with the Tseitin-style translation implemented in bc2cnf for running the CNF-level methods.

The experimental results presented in tabular form in **P6** compare the number of moves made by WalkSAT and a straightforward prototype implementation of BC SLS with $q = 0$. The noise parameter p for making non-greedy moves was set to the default value of 50% for both WalkSAT and BC SLS. Here we will provide another view to these results. A comparison of the number of moves is shown in Figure 23. For each data point (x, y), x represents the number of moves made by WalkSAT on the ith best run and

[3] Available at `http://www.tcs.tkk.fi/~mjj/benchmarks/`.

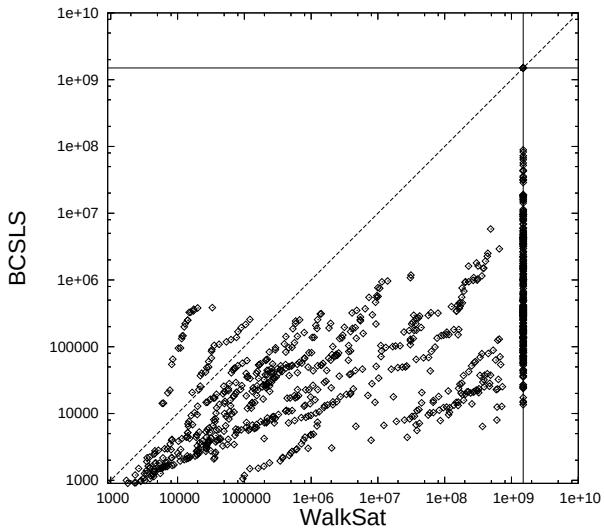


Figure 23: BC SLS and WalkSAT: Comparison of number of moves

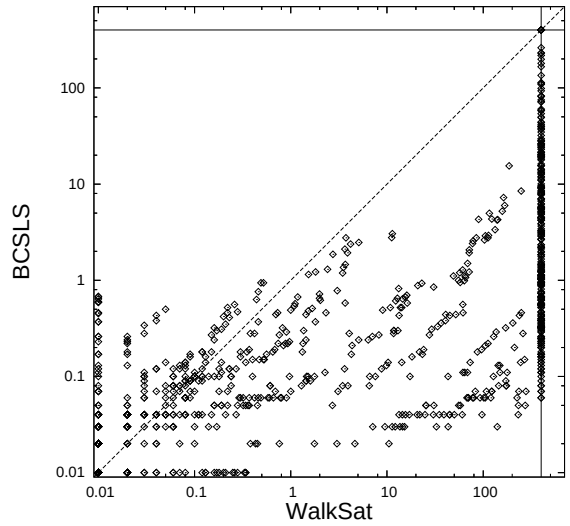


Figure 24: BC SLS and WalkSAT: Comparison of running times

y the number of moves by BC SLS on the ith best run. The data is gathered over 21 runs (that is, $i = 1, \ldots, 21$) without restarts (MAXTRIES($\mathcal{C}^\alpha = 1$)) and with MAXMOVES($\mathcal{C}^\alpha$) $= 10^9$ for each run. The horizontal and vertical lines at $5 \cdot 10^9$ represent runs where a satisfying truth assignment was not found with 10^9 moves. We notice that, quite generally, BC SLS needs up to multiple orders of magnitude less moves than WalkSAT.

Compared to moves, it is practically more relevant to compare the running times of different methods. While such comparison is not provided in **P6**, we provide one here (Figure 24). The running times for WalkSAT are from the fast implementation of the method found in the UBCSAT SLS solver [232]. The running times for BC SLS are from a current implementation by T. Junttila in which the computation of heuristics (justification front, cone, interest set) is incremental. The time out was set to 5 minutes, with the timed out runs on the horizontal and vertical lines. The data points represent the running times in a similar fashion as in Figure 23 for moves. We notice that, while WalkSAT and BC SLS seem to be equally competitive on easier instances, BC SLS shows better scalability as instances get harder. Especially noticeable is the large number of timed out runs for WalkSAT.

Experiments with Adaptive Noise Strategies for BC SLS

Considering CNF-level SLS methods for SAT, it has been noticed that SLS performance can vary critically depending on the chosen noise setting [114], and the optimal noise setting can vary from instance to instance and within families of similar instances. The same phenomenon is present also in BC SLS. This observation has led to the development of an *adaptive noise mechanism* for CNF-level SLS in the solver AdaptNovelty+ [114], dismissing the requirement of a pre-tuned noise parameter. This idea has been successfully applied in other SLS solvers as well [160]. In **P7** we consider adaptive noise strategies for BC SLS.

Following the general idea presented in [114], a generic adaptive noise mechanism for BC SLS is presented as Algorithm 4. Starting from $p = 0$,

Algorithm 4 Generic Adaptive Noise Mechanism

```
   p: noise (initially p = 0)
   adapt_score: score at latest noise change
   adapt_step: step of latest noise change
1: if score < adapt_score then                          %% noise decrease
2:     p := p − (φ/2) · p
3:     adapt_step := step
4:     adapt_score := score
5: else                                                 %% noise increase
6:     if (step − adapt_step) > WaitingPeriod() then
7:         p := p + φ · (1 − p)
8:         adapt_step := step
9:         adapt_score := score
```

the noise setting is tuned during search based on the development of the objective function value. *Every time* the objective function value is improved, noise is decreased according to line 2. If no improvement in the objective function value has been observed during the last WaitingPeriod() steps, the noise is increased according to line 7, where $\phi \in]0, 1[$ controls the relative amount of noise increase. Each time the noise setting is changed, the current objective function value is then stored for the next comparison.

Hoos [114] suggests, reporting generally good performance, to use $\phi = \frac{1}{5}$ and the *static* function $\theta \cdot C$ for WaitingPeriod(), where $\theta = \frac{1}{6}$ is a constant and C denotes the number of clauses in the CNF instance at hand. These parameter values have been applied also in other CNF-level SLS solvers [160].

For BC SLS, we fix ϕ accordingly to $\frac{1}{5}$. In other words, we focus on investigating the effect of applying different waiting periods for noise increases in the context of BC SLS. As the first step, in **P7** we investigate using as WaitingPeriod() a static linear function defined by the number of unconstrained gates multiplied by a constant θ. In fact, opposed to reported experience with CNF-level SLS [114], it turns out that, for BC SLS, the value $\theta = \frac{1}{6}$ is too large: by decreasing θ, we can increase the performance of BC SLS. As shown in **P7**, by decreasing θ to $\frac{1}{24}$ we witness an evident overall gain in performance against $\theta = \frac{1}{6}$, and again by decreasing θ from $\frac{1}{24}$ to $\frac{1}{96}$. However, we noticed that changing the overall scheme in the original adaptive noise mechanism leads to even better performance for BC SLS. In the novel scheme that we call *rapidly increasing*, when the waiting period is exceeded, the noise level is increased after *each* step until we see the first one-step improvement in the objective function. This can be implemented by removing line 8 in Algorithm 4.

While the reader is referred to **P7** for more detailed experimental results on variations of adaptive noise for BC SLS, we will here provide a comparison of the running times of AdaptNovelty+ implementation in UBCSAT [232] and BC SLS with $\theta = \frac{1}{96}$ using the rapidly increasing noise strategy.

The results are encouraging. Again, here we will provide another view

to these results, complementing the results presented in Table 1 in **P7**. Although making moves is slower in the current implementation of BC SLS (around 300.000 moves per second on average) than in AdaptNovelty+ (around 2.5 million per second), BC SLS is very much competitive in running times on these instances (see Figure 26) as less moves are usually needed for finding a solution (as shown in Figure 25).

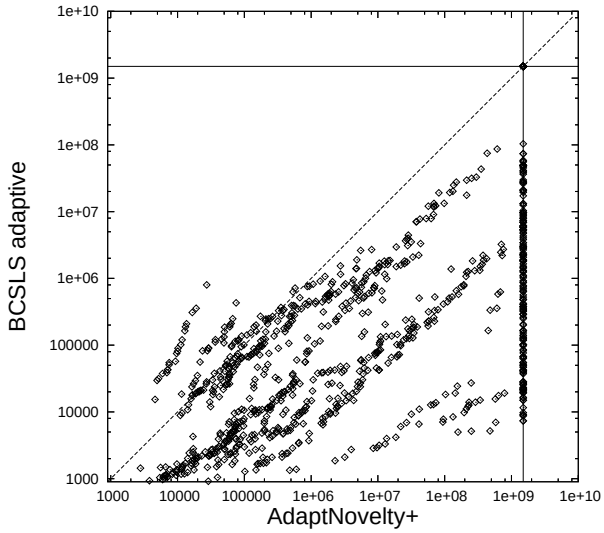


Figure 25: Adaptive BC SLS and AdaptNovelty+: Comparison of number of moves

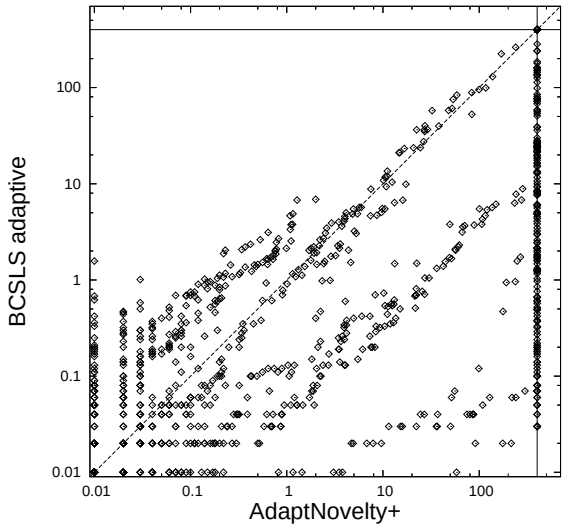


Figure 26: Adaptive BC SLS and AdaptNovelty+: Comparison of running times

Dynamic Waiting Periods

With experimental evidence provided in **P7**, we notice that by employing the adaptive noise mechanism based on static waiting periods, we may have only changed the problem of finding the optimal static noise level parameter p into the problem of finding an instance-specific optimal value for θ. This motivates us to consider, opposed to a static waiting period controlled by the addition parameter θ, *dynamic waiting periods* based on the state of search, with the possibility of dismissing the otherwise required constant θ.

We consider in **P7** two dynamic alternatives for adjusting the waiting period: WAITINGPERIOD() $=$ $\mathsf{jfront}(\mathcal{C}^\alpha, \tau)$ (the size of the current justification frontier), and WAITINGPERIOD() $=$ $\mathsf{interest}(\mathcal{C}^\alpha, \tau)$ (the size of the current interest set). The intuition behind using the size of the justification frontier is that, since the gate at each step is selected from the justification frontier, the size of the frontier gives us an estimate on the number of possible greedy moves in order to improve the objective function value before increasing the possibility of non-greedy moves (increasing noise). On the other hand, the size of the interest set is precisely the objective function value. Intuitively, the greater the objective function value is, the farther we are from a solution, and thus more effort is allowed on finding a good greedy move.

While we refer to **P7** for details, interestingly enough these dynamic waiting periods result in comparable performance as with the small static waiting period of $\theta = \frac{1}{96}$ on the considered set of BMC Boolean circuit benchmarks.

5 CONCLUSIONS

This work aims at contributing to the understanding of the relationship of structural aspects of real-world SAT instances and the effectiveness of search-based SAT solving techniques for such problems.

A major contribution of this work addresses the effect of structure-based branching heuristics on the efficiency of typical complete SAT solver techniques based on DPLL and clause learning. This topic is well-motivated by the fact that, while techniques such as novel decision heuristics and clause learning have been the focus of much attention, the structural properties underlying CNF encodings of real-world problems have not been extensively studied from the view point of branching restrictions. Although branching plays a key role in search for satisfiability, there still is no general consensus on what type of structural properties (if any) reflect variables with high importance with respect to efficiency of search, and how such knowledge could be exploited in making SAT solvers more robust.

With propositional proof complexity as the framework, we show in **P3** that when branching is statically restricted to input variables in clause learning DPLL, the resulting underlying proof system weakens considerably; input-restricted branching clause learning DPLL and basic DPLL are polynomially incomparable. This holds even when input-restricted branching clause learning DPLL is allowed unlimited restarts and the ability to branch on variables with already assigned values. This also implies that all implementations of clause learning DPLL, even with optimal heuristics, have the potential of suffering a notable efficiency decrease when input-restricted branching is applied.

The experimental results of **P2** confirm that, in general, input-restricted branching can cause a notable loss of robustness in a clause learning SAT solver. For example, input-restricted branching results in longer conflict clauses on the average, which in itself makes clause learning less effective and can also hinder the overall efficiency of the solver. However, by relaxing the branching restriction in a systematic fashion, branching can in fact be restricted quite heavily without making a clause learning solver notably less efficient. Moreover, the choice of the structural property on which such a relaxation is based on does make a difference.

In addition to the static branching restriction based on input variables, in **P4** we study the proof complexity theoretical effect of dynamically restricting branching in DPLL and clause learning based on variations of top-down branching. The result is a relative efficiency hierarchy for variations of circuit-level DPLL and CL. Perhaps the most surprising result obtained in **P4** is that CL using unlimited restarts with justification restricted decision heuristics cannot even simulate the top-down restricted variant of DPLL. Thus, although the idea of eagerly and locally justifying the values of currently unjustified constraints is an intuitively appealing one, it can lead to dramatic losses in the best-case efficiency of a structure-aware clause learning SAT solver.

In connection with structure-based branching in SAT, the work in **P5** introduces the Extended ASP Tableaux proof system in the context of answer set programming. Exploiting known results on the power of the extended resolution proof system for CNF SAT, the extension rule of Extended ASP

Tableaux allows for adding redundant structure to ASP instances. Through the extension rule ASP solvers, which are closely related to DPLL-based SAT solvers, may branch on the substructures introduced to programs using the extension rule. A major motivation behind **P5** is to continue bridging the gap between search-based methods for ASP and SAT. In addition to theoretical observations on Extended ASP Tableaux, the effect of adding redundant structure to ASP instances on the efficiency of ASP solvers is experimentally studied.

Another aspect of structure-based search techniques for SAT addressed in this work is the challenge of improving stochastic local search techniques for structured SAT instances and, in particular, developing new techniques that exploit variable dependencies in the context of SLS. A novel non-clausal SLS method, BC SLS, for structured SAT instances is developed in **P6**, with the aim of lifting the performance of local search SAT solving especially on instances stemming from real-world problem domains. Motivated by justification frontier based heuristics applied in complete circuit-level SAT solvers in electronic design automation, the presented SLS method looks for a justification for the Boolean circuit instead of focusing on finding a satisfying truth assignment. The idea is to be able to drive local search more top-down in the overall structure of the circuit rather than in a bottom-up mode as is done in local search techniques focusing on input variables. The justification frontier based SLS heuristics enables observability don't cares to be exploited in SLS and offers an early stopping criterion for ending the search.

In **P7** variants of BC SLS are analyzed with respect to the PAC property, highlighting that PAC can be achieved while still keeping the search focused with the justification frontier based heuristics. Furthermore, adaptive noise mechanisms aimed specifically for BC SLS are considered and experimented with. A current implementation of BC SLS can outperform typical CNF-level SLS methods such as WalkSAT and AdaptNovelty+ in running times and in the number of moves up to multiple magnitudes of difference on real-world BMC circuit instances.

As a third point of view to structure in SAT, the work in **P1** addresses the problem of generating hard satisfiable SAT instances for both DPLL-based and local search solvers by introducing the regular XORSAT model. Motivated by the good expansion properties of random regular graphs, the model combines regularity and randomness in the underlying constraint graphs of the instances in order to generate exceptionally difficult CNF SAT instances. Techniques for applying XORSAT instances specifically for benchmarking equivalence reasoning techniques in SAT solvers are also developed.

5.1 Topics for Further Work

We conclude by discussing some possibilities for further work related to the topic of this thesis.

The experimental results on restricting branching in clause learning SAT solvers imply that, in general, many branching restrictions based on natural structural properties of circuit gates do not give notable gains. Can such, relatively small branching restrictions still be found, for example, by further analyzing combinations of structural properties? Moreover, if static branch-

ing restrictions do not increase efficiency on their own, as it seems by the experimental results in this work, one could investigate more complex, dynamic branching restrictions for structural problems; an interesting question is whether solver efficiency could be increased by developing structure-aware branching restriction techniques that act *dynamically* in cooperation with clause learning, especially for Boolean circuit level SAT solvers such as [230].

Another relevant direction of further study is the possibility of restricting branching based on the known structure of known/novel CNF encodings of more general Boolean constraints. A step into this direction is taken in a recent work [173] which studies this possibility in the special case of At-Most-One cardinality constraints. Could intelligent domain specific branching restrictions result in practical gains, for instance, in satisfiability modulo theories (SMT) solvers, where the constraint structure is more complicated than that in the pure SAT-based approaches?

Questions related to a more exact proof complexity theoretic characterization of the power of clause learning solvers compared to the results presented in [33] are still open: what is exactly the relative power of clause learning DPLL without restarts? Can it polynomially simulate RES without further relaxations? Within the results achieved in this work, a positive answer to this question would further clarify, for example, the hierarchy shown in Figure 12. Related to the power of clause learning and DPLL-based solvers, additional interesting questions to investigate deal with the effect of applying additional search techniques, such as polynomial time propagation mechanisms other than unit propagation, which may raise the power of clause learning even beyond resolution. For instance, specialized symmetry exploiting techniques during search [159, 203] intuitively bring the resulting solvers closer to the power of *symmetric resolution* [146, 236]; applying *formula caching* [30] in DPLL serves as another interesting example.

The question of whether the underlying power of extended resolution, based on redundancy, could be harnessed in even a limited way for achieving practical gains in SAT solvers is an intriguing one. To the best of our knowledge, by now extended resolution has been applied in the context of SAT solving mainly in compactly representing BDD construction [221, 134] as extended resolution proofs. Would it be possible to harness ideas related to extended resolution, for example, for creating problem domain specific automated reformulation techniques for SAT encodings with practical relevance?

Considering structure-based stochastic local search for SAT, there are many possibilities for further study within and related to the BC SLS method developed in this work. One aspect is to study the applicability of justification-based local search solving for logical combinations of more general constraints, arising from constraint programming (CP) and SMT, for instance. Another aspect would be to perform an in-depth study of the applicability of different structure-based SLS heuristics on instances with different structural properties. Furthermore, for testing the limits of SLS methods on real-world problem instances, effective restart strategies for SLS should be investigated. As a unifying view to the development of more efficient complete and SLS solvers, the potential of devising robust *hybrid methods*, incorporating techniques from both DPLL-based and SLS methods, should be studied further.

Taking a general view to constraint satisfaction, the approaches developed

in the closely related fields of SAT, SMT, ASP, and CP have all particular advantages when solving real-world problems. A unifying understanding of what is needed for achieving efficient and robust constraint satisfaction methods for different structured real-world problem domains remains a major challenge. The right level of abstraction in the core language on which the search for satisfaction is performed—which today varies from high-level CP languages, where vast numbers of global constraints are available, to the very low-level language of CNF SAT—plays a key role in enabling the development of efficient constraint satisfaction solvers. This motivates further work in understanding the relationships between these languages, and, in particular, further investigations into structure-preserving decompositions of high-level constraint models using low-level primitive constraint types.

REFERENCES

[1] Parosh Aziz Abdulla, Per Bjesse, and Niklas Eén. Symbolic reachability analysis based on SAT-solvers. In Susanne Graf and Michael I. Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2000.

[2] Dimitris Achlioptas, Paul Beame, and Michael Molloy. Exponential bounds for DPLL below the satisfiability threshold. In J. Ian Munro, editor, *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 139–140. SIAM, 2004.

[3] Dimitris Achlioptas, Paul Beame, and Michael S. O. Molloy. A sharp threshold in proof complexity yields lower bounds for satisfiability search. *Journal of Computer and System Sciences*, 68(2):238–268, 2004.

[4] Dimitris Achlioptas, Carla P. Gomes, Henry A. Kautz, and Bart Selman. Generating satisfiable problem instances. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'00)*, pages 256–261. AAAI Press, 2000.

[5] Dimitris Achlioptas, Haixia Jia, and Cristopher Moore. Hiding truth assignments: Two are better than one. *Journal of Artificial Intelligence Research*, 24:623–639, 2005.

[6] Michael Alekhnovich. Mutilated chessboard problem is exponentially hard for resolution. *Theoretical Computer Science*, 310(1–3):513–525, 2004.

[7] Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson. Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *Journal of Automated Reasoning*, 35(1–3):51–72, 2005.

[8] Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC'02)*, pages 448–456. ACM, 2002.

[9] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[10] Noga Alon and Vitali D. Milman. λ_1, isoperimetric inequalities for graphs and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.

[11] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. Shatter: efficient symmetry-breaking for boolean satisfiability. In *Proceedings of the 40th Design Automation Conference (DAC'03)*, pages 836–839. ACM, 2003.

[12] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. MINCE: A static global variable-ordering heuristic for SAT search and BDD manipulation. *Journal of Universal Computer Science*, 10(12):1562–1596, 2004.

[13] Anbulagan, Duc Nghia Pham, John K. Slaney, and Abdul Sattar. Old resolution meets modern SLS. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, pages 354–359. AAAI Press / The MIT Press, 2005.

[14] Anbulagan and John Slaney. Lookahead saturation with restriction for SAT. In Peter van Beek, editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *Lecture Notes in Computer Science*, pages 727–731. Springer, 2005.

[15] Henrik Reif Andersen and Henrik Hulgaard. Boolean expression diagrams. *Information and Computation*, 179(2):194–212, 2002.

[16] Christian Anger, Martin Gebser, Tomi Janhunen, and Torsten Schaub. What's a head without a body? In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, pages 769–770. IOS Press, 2006.

[17] Christian Anger, Martin Gebser, Thomas Linke, André Neumann, and Torsten Schaub. The nomore++ approach to answer set solving. In Geoff Sutcliffe and Andrei Voronkov, editors, *Proceeding of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2005.

[18] Noriko H. Arai, Toniann Pitassi, and Alasdair Urquhart. The complexity of analytic tableaux. In *Proceedings on 33th Annual ACM Symposium on Theory of Computing (STOC'01)*, pages 356–363. ACM, 2001.

[19] Alessandro Armando, Luca Compagna, and Pierre Ganty. SAT-based model-checking of security protocols using planning graph analysis. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *Proceedings of the 2003 International Symposium of Formal Methods Europe (FME'03)*, volume 2805 of *Lecture Notes in Computer Science*, pages 875–893. Springer, 2003.

[20] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. In Antti Valmari, editor, *Proceedings of the 13th International SPIN Workshop on Model Checking Software*, volume 3925 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2006.

[21] Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In Mark Wallace, editor, *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004.

[22] Fachiem Bacchus. Enhancing Davis-Putnam with extended binary clause reasoning. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'02)*, pages 613–619. AAAI Press, 2002.

[23] Fahiem Bacchus. GAC via unit propagation. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 2007.

[24] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS'03)*, pages 340–351. IEEE Computer Society, 2003.

[25] Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03), Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 2004.

[26] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[27] Wolfgang Barthel, Alexander K. Hartmann, Michele Leone, Federico Ricci-Tersenghi, Martin Weight, and Riccardo Zecchina. Hiding solutions in random satisfiability problems: A statistical mechanics approach. *Physical Review Letters*, 18:188701, 2002.

[28] Peter Baumgartner and Fabio Massacci. The taming of the (X)OR. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proceedings of the 1st International Conference on Computational Logic (CL'00)*, volume 1861 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2000.

[29] Paul Beame, Joseph C. Culberson, David G. Mitchell, and Cristopher Moore. The resolution complexity of random graph k-colorability. *Discrete Applied Mathematics*, 153(1–3):25–47, 2005.

[30] Paul Beame, Russell Impagliazzo, Toniann Pitassi, and Nathan Segerlind. Memoization and DPLL: Formula caching proof systems. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity (CCC'03)*, pages 225–236. IEEE Computer Society, 2003.

[31] Paul Beame, Russell Impagliazzo, and Ashish Sabharwal. The resolution complexity of independent sets and vertex covers in random graphs. *Computational Complexity*, 16(3):245–297, 2007.

[32] Paul Beame, Richard M. Karp, Toniann Pitassi, and Michael E. Saks. The efficiency of resolution and Davis–Putnam procedures. *SIAM Journal on Computing*, 31(4):1048–1075, 2002.

[33] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[34] Paul Beame and Toniann Pitassi. Propositional proof complexity: Past, present, and future. *Bulletin of the EATCS*, 65:66–89, 1998.

[35] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12(1–2):53–87, 1994.

[36] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

[37] Armin Biere. Adaptive restart strategies for conflict driven sat solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer, 2008.

[38] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of the 36th Conference on Design Automation (DAC'99)*, pages 317–320. ACM Press, 1999.

[39] Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5:5), 2006.

[40] Armin Biere and Wolfgang Kunz. SAT and ATPG: Boolean engines for formal hardware verification. In *Proceedings of the 20th IEEE/ACM International Conference on Computer Aided Design (ICCAD'02)*, pages 782–785. IEEE Computer Society, 2002.

[41] Per Bjesse and Arne Borälv. DAG-aware circuit compression for formal verification. In *Proceedings of the 2004 International Conference on Computer-Aided Design (ICCAD'04)*, pages 42–49. IEEE Computer Society / ACM, 2004.

[42] Manuel Blum, Richard M. Karp, Oliver Vornberger, Christos H. Papadimitriou, and Mihalis Yannakakis. The complexity of testing whether a graph is a superconcentrator. *Information Processing Letters*, 13(4-5):164–167, 1981.

[43] Bela Bollobás. The isoperimetric number of random regular graphs. *European Journal of Combinatorics*, 9(3):241–244, 1988.

[44] Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys*, 38(4):Article 12, 2006.

[45] Yacine Boufkhad, Olivier Dubois, Yannet Interian, and Bart Selman. Regular random k-SAT: Properties of balanced formulas. *Journal of Automated Reasoning*, 35(1–3):181–200, 2005.

[46] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani. MathSAT: Tight integration of SAT and mathematical decision procedures. *Journal of Automated Reasoning*, 35(1–3):265–293, 2005.

[47] Ronen I. Brafman. A simplifier for propositional formulas with many binary clauses. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(1):52–59, 2004.

[48] Stefan Brass and Jürgen Dix. Characterizations of the stable semantics by partial evaluation. In V. Wiktor Marek and Anil Nerode, editors, *Proceedings of the 3rd International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'95)*, volume 928 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 1995.

[49] Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures and Algorithms*, 27(2):201–226, 2005.

[50] Gerhard Brewka, Ilkka Niemelä, and Mirosław Truszczyński. *Nonmonotonic reasoning*, chapter 6, pages 239–284. In van Harmelen et al. [238], 2008.

[51] Randy E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[52] Josh Buresh-Oppenheim and Toniann Pitassi. The complexity of resolution refinements. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 138–. IEEE Computer Society, 2003.

[53] Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.

[54] Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artificial Intelligence*, 147(1–2):85–117, 2003.

[55] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, 1988.

[56] Keith L. Clark. Negation as failure. In *Readings in nonmonotonic reasoning*, pages 311–325. Morgan Kaufmann Publishers, 1987.

[57] Edmund Clarke, Armin Biere, Richard Raimi, and Yushan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[58] Stephen A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC'71)*, pages 151–158. ACM, 1971.

[59] Stephen A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, 1976.

[60] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

[61] James M. Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'94)*, pages 1092–1097. AAAI Press, 1994.

[62] Stefan Dantchev and Soren Riis. "Planar" tautologies hard for resolution. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS'01)*, pages 220–229. IEEE Computer Society, 2001.

[63] Adnan Darwiche. Decomposable negation normal form. *Journal of ACM*, 48(4):608–647, 2001.

[64] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[65] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[66] Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion attacks on secure hash functions using SAT solvers. In João Marques-Silva and Karem A. Sakallah, editors, *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*. Springer, 2007.

[67] Thierry Boy de la Tour. An optimality result for clause form translation. *Journal of Symbolic Computation*, 14:283–301, 1992.

[68] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[69] Bistra Dilkina, Carla P. Gomes, and Ashish Sabharwal. Tradeoffs in the complexity of backdoor detection. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and*

Practice of Constraint Programming (CP'07), volume 4741 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2007.

[70] Rolf Drechsler and Bernd Becker. *Binary Decision Diagrams – Theory and Implementation*. Kluwer Academic Publishers, 1998.

[71] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[72] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03), Revised Selected Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[73] Tobias Eibach, Enrico Pilz, and Gunnar Völkel. Attacking bivium using SAT solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 63–76. Springer, 2008.

[74] Francois Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[75] Lei Fang and Michael S. Hsiao. A new hybrid solution to boost SAT solver performance. In Rudy Lauwereins and Jan Madsen, editors, *Proceedings of the 2007 Design, Automation and Test in Europe Conference and Exposition (DATE'07)*, pages 1307–1313. ACM, 2007.

[76] Joel Friedman. On the second eigenvalue and random walks in random d-regular graphs. *Combinatorica*, 11(4):331–362, 1991.

[77] Zhaohui Fu and Sharad Malik. Extracting logic circuit description from conjunctive normal form descriptions. In *Proceedings of the IEEE/ACM 20th International Conference on VLSI Design*, pages 37–42. IEEE Computer Society, 2007.

[78] Malay K. Ganai and Andreas Kuehlmann. On-the-fly compression of logical circuits. In *Informal Proceedings of the 9th IEEE/ACM International Workshop on Logic Synthesis (IWLS'00)*, 2000. Available at `http://citeseer.ist.psu.edu/ganai00fly.html`.

[79] Malay K. Ganai, Lintao Zhang, Pranav Ashar, Aarti Gupta, and Sharad Malik. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *Proceedings of the 39th Design Automation Conference, (DAC'02)*, pages 747–750. ACM, 2002.

[80] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[81] Juan P. Garrahan and Mark E.J. Newman. Glassiness and constrained dynamics of short-range non-disordered spin model. *Physical Review E*, 62:7670–7678, 2000.

[82] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set solving. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 286–392, 2007.

[83] Martin Gebser and Torsten Schaub. Characterizing ASP inferences by unit propagation. In E.Giunchiglia, V. Marek, D. Mitchell, and E. Ternovska, editors, *Proceedings of the ICLP'06 Workshop on Search and Logic: Answer Set Programming and SAT*, pages 41–56, 2006.

[84] Martin Gebser and Torsten Schaub. Tableau calculi for answer set programming. In Sandro Etalle and Miroslaw Truszczynski, editors, *Proceedings of the 22nd International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*, pages 11–25. Springer, 2006.

[85] Martin Gebser and Torsten Schaub. Generic tableaux for answer set programming. In Verónica Dahl and Ilkka Niemelä, editors, *Proceedings of the 23rd International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 2007.

[86] Michael Gelfond. *Answer sets*, chapter 7, pages 285–316. In van Harmelen et al. [238], 2008.

[87] Michael Gelfond and Nicola Leone. Logic programming and knowledge representation – the A-Prolog perspective. *Artificial Intelligence*, 138(1–2):3–38, 2002.

[88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP'88)*, pages 1070–1080. MIT Press, 1988.

[89] Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*, 36(4):345–377, 2006.

[90] Enrico Giunchiglia and Marco Maratea. On the relation between answer set and SAT procedures (or, between CMODELS and SMODELS). In Maurizio Gabbrielli and Gopal Gupta, editors, *Proceedings of the 21st International Conference on Logic Programming*

(ICLP'05), volume 3668 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2005.

[91] Enrico Giunchiglia, Marco Maratea, and Armando Tacchella. Dependent and independent variables in propositional satisfiability. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *Proceedings of the European Conference on Logics in Artificial Intelligence (JELIA'02)*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 296–307. Springer, 2002.

[92] Enrico Giunchiglia, Alessandro Massarotto, and Roberto Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In B.G. Buchanan C. Rich, J. Mostow and R. Uthurusamy, editors, *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 948–953. AAAI Press, 1998.

[93] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Quantifier structure in search-based procedures for QBFs. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 26(3):497–507, 2007.

[94] Fausto Giunchiglia and Roberto Sebastiani. Building decision procedures for modal logics from propositional decision procedures: the case study of modal K. In Michael A. McRobbie and John K. Slaney, editors, *In Proceedings of 13th International Conference on Automated Deduction (CADE'96)*, volume 1104 of *Lecture Notes in Computer Science*, pages 583–597, 1996.

[95] Andreas Goerdt. Regular resolution versus unrestricted resolution. *SIAM Journal on Computing*, 22(4):661–683, 1993.

[96] Vibhav Gogate and Rina Dechter. Approximate counting by sampling the backtrack-free search space. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 198–203. AAAI Press, 2007.

[97] Evgueni Goldberg and Yakov Novikov. Berkmin: A fast and robust SAT-solver. In *Proceedings of the 2002 Design, Automation and Test in Europe Conference (DATE'02)*, pages 142–149. IEEE Computer Society, 2002.

[98] Carla P. Gomes, Henry A. Kautz, Ashish Sabharwal, and Bart Selman. *Satisfiability solvers*, chapter 2, pages 89–134. In van Harmelen et al. [238], 2008.

[99] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting: A new strategy for obtaining good bounds. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 54–61. AAAI Press, 2006.

[100] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1–2):67–100, 2000.

[101] Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In B.G. Buchanan C. Rich, J. Mostow and R. Uthurusamy, editors, *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98)*, pages 431–437. AAAI Press, 1998.

[102] Carla P. Gomes, Bart Selman, and Ryan Williams. On the connections between backdoors and heavy-tails on combinatorial search. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, 2003.

[103] Alban Grastien, Anbulagan, Jussi Rintanen, and Elena Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 305–310. AAAI Press, 2007.

[104] Éric Grégoire, Richard Ostrowski, Bertrand Mazure, and Lakhdar Sais. Automatic extraction of functional dependencies. In Holger H. Hoos and David G. Mitchell, editors, *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 122–132. Springer, 2005.

[105] Jan Friso Groote and Hans Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. *Discrete Applied Mathematics*, 130(2):157–171, 2003.

[106] Djamal Habet, Chu Min Li, Laure Devendeville, and Michel Vasquez. A hybrid approach for SAT. In Pascal Van Hentenryck, editor, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 172–184. Springer, 2002.

[107] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, 1985.

[108] Keijo Heljanko. Bounded reachability checking with process semantics. In Kim Guldstrand Larsen and Mogens Nielsen, editors, *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2001.

[109] Marijn Heule, Mark Dufour, Joris van Zwieten, and Hans van Maaren. March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver. In Holger H. Hoos and David G. Mitchell, editors, *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2005.

[110] Marijn Heule and Hans van Maaren. Aligning CNF- and equivalence-reasoning. In Holger H. Hoos and David G. Mitchell, editors, *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2005.

[111] Edward A. Hirsch and Arist Kojevnikov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43(1):91–111, 2005.

[112] John N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15(3):359–383, 1995.

[113] Holger H. Hoos. On the run-time behaviour of stochastic local search algorithms for sat. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI'99)*, pages 661–666, 1999.

[114] Holger H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of the 18th National Conference on Artificial intelligence (AAAI'02)*, pages 655–660. AAAI Press, 2002.

[115] Holger H. Hoos and Thomas Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[116] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[117] Jinbo Huang. The effect of restarts on the efficiency of clause learning. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2318–2323. AAAI Press, 2007.

[118] Jinbo Huang and Adnan Darwiche. A structure-based variable ordering heuristic for SAT. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1167–1172. Morgan Kaufmann, 2003.

[119] Madhu K. Iyer, Ganapathy Parthasarathy, and Kwang-Ting Cheng. SATORI—a fast sequential SAT engine for circuits. In *Proceedings of the International Conference on Computer Aided Design (ICCAD'03)*, pages 320–325. IEEE Computer Society, 2003.

[120] Paul Jackson and Daniel Sheridan. Clause form conversions for Boolean circuits. In Holger H. Hoos and David G. Mitchell, editors, *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2005.

[121] Tomi Janhunen. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics*, 16(1-2):35–86, 2006.

[122] Matti Järvisalo. Further investigations into regular XORSAT. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 1873–1874. AAAI Press, 2006.

[123] Matti Järvisalo. The effect of structural branching on the efficiency of clause learning SAT solving. In *14th RCRA Workshop: Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, 2007. See `http://pst.istc.cnr.it/RCRA07/program.html`.

[124] Matti Järvisalo and Tommi Junttila. Limitations of restricted branching in clause learning. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2007.

[125] Matti Järvisalo, Tommi Junttila, and Ilkka Niemelä. Unrestricted vs restricted cut in a tableau method for Boolean circuits. *Annals of Mathematics and Artificial Intelligence*, 44(4):373–399, 2005.

[126] Matti Järvisalo and Emilia Oikarinen. Extended ASP tableaux and rule redundancy in normal logic programs. In Verónica Dahl and Ilkka Niemelä, editors, *Proceedings of the 23rd International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2007.

[127] Haixia Jia, Cristopher Moore, and Bart Selman. From spin glasses to hard satisfiable formulas. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2005.

[128] Haixia Jia, Cristopher Moore, and Doug Strain. Generating hard satisfiable formulas by hiding solutions deceptively. In *Proceedings of the 20h National Conference on Artificial Intelligence (AAAI'05)*, pages 384–389. AAAI Press, 2005.

[129] Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. Counting models using connected components. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*, pages 157–162. AAAI Press / The MIT Press, 2000.

[130] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In Benjamin K. Kuipers and Bonnie Webber, editors, *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208. AAAI Press, 1997.

[131] Tommi Junttila. A file format for constrained boolean circuits. Available at `http://www.tcs.hut.fi/~tjunttil/bcsat/`.

[132] Tommi A. Junttila and Ilkka Niemelä. Towards an efficient tableau method for boolean circuit satisfiability checking. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proceedings of the 1st International Conference on Computational Logic (CL'00)*, volume 1861 of *Lecture Notes in Computer Science*, pages 553–567. Springer, 2000.

[133] Toni Jussila, Keijo Heljanko, and Ilkka Niemelä. BMC via on-the-fly determinization. *International Journal on Software Tools for Technology Transfer*, 7(2):89–101, 2005.

[134] Toni Jussila, Carsten Sinz, and Armin Biere. Extended resolution proofs for symbolic SAT solving with quantification. In Armin Biere and Carla P. Gomes, editors, *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT 2006)*, volume 4121 of *Lecture Notes in Computer Science*, pages 54–60. Springer, 2006.

[135] Petteri Kaski. Barriers and local minima in energy landscapes of stochastic local search. *CoRR*, abs/cs/0611103, 2006. `http://arxiv.org/abs/cs/0611103`.

[136] Henry A. Kautz. Deconstructing planning as satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 1524–1526. AAAI Press, 2006.

[137] Henry A. Kautz, David McAllester, and Bart Selman. Exploiting variable dependency in local search. In *Poster Sessions of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, 1997. Available at `http://www.cs.cornell.edu/home/selman/papers-ftp/97.ijcai.dagsat.ps`.

[138] Henry A. Kautz, Yongshao Ruan, Dimitris Achlioptas, Carla P. Gomes, Bart Selman, and Mark E. Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 351–358. Morgan Kaufmann, 2001.

[139] Henry A. Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363. John Wiley and Sons, 1992.

[140] Henry A. Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96)*, pages 1194–1201. AAAI Press, 1996.

[141] Henry A. Kautz and Bart Selman. The state of SAT. *Discrete Applied Mathematics*, 155(12):1514–1524, 2007.

[142] Scott Kirkpatrick and Bart Selman. Critical behaviour in the satisfiability of random Boolean expressions. *Science*, 264:1297–1301, 1994.

[143] Alexander V. Kostochka and Leonid S. Melnikov. On a lower bound for the isoperimetric number of cubic graphs. In *Probabilistic Methods in Discrete Mathematics*, volume 1 of *Progress in Pure and Applied Discrete Mathematics*, pages 251–265. VSP, 1993.

[144] Jan Krajíček. *Bounded arithmetic, propositional logic, and complexity theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 1995.

[145] Jan Krajíček. An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams. *Journal of Symbolic Logic*, 73(1):227–237, 2008.

[146] Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, 1985.

[147] Lukas Kroc, Ashish Sabharwal, and Bart Selman. Leveraging belief propagation, backtrack search, and statistics for model counting. In *Proceedings of International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR'08)*, volume 5015 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2008.

[148] Daniel Kroening, Edmund Clarke, and Karen Yorav. Behavioral consistency of C and Verilog programs using bounded model checking. In *Proceedings of the 40th Conference on Design Automation (DAC'03)*, pages 368–371. ACM Press, 2003.

[149] Andreas Kuehlmann, Malay K. Ganai, and Viresh Paruthi. Circuit-based Boolean reasoning. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 232–237. ACM, 2001.

[150] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1377–1394, 2002.

[151] Tracy Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):6–22, 1992.

[152] Timo Latvala, Armin Biere, Keijo Heljanko, and Tommi A. Junttila. Simple bounded LTL model checking. In Alan J. Hu and Andrew K. Martin, editors, *Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD'04)*, volume 3312 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2004.

[153] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[154] Florian Letombe and João Marques-Silva. Improvements to hybrid incremental SAT algorithms. In Hans Kleine Büning and Xishun Zhao, editors, *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 168–181. Springer, 2008.

[155] Reinhold Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In Uwe Egly and Christian G. Fermüller, editors, *Proceedins of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'02)*, volume 2381 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2002.

[156] Chu Min Li. A constraint-based approach to narrow search trees for satisfiability. *Information Processing Letters*, 71(2):75–80, 1999.

[157] Chu Min Li. Equivalent literal propagation in Davis-Putnam procedure. *Discrete Applied Mathematics*, 130(2):251–276, 2003.

[158] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In M. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371. Morgan Kaufmann, 1997.

[159] Chu Min Li, Bernard Jurkowiak, and Paul W. Purdom Jr. Integrating symmetry breaking into a DLL procedure. In *Proceedings of the 5th Symposium on Theory and Applications of Satisfiability Testing (SAT'02)*, 2002.

[160] Chu Min Li, Wanxia Wei, and Harry Zhang. Combining adaptive noise and look-ahead in local search for SAT. In João Marques-Silva and Karem A. Sakallah, editors, *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2007.

[161] Xiao Yu Li, Matthias F. M. Stallmann, and Franc Brglez. A local search SAT solver using an effective switching strategy and an efficient unit propagation. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03), Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2004.

[162] Yuliya Lierler and Marco Maratea. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Computer Science*, pages 346–350. Springer, 2004.

[163] Vladimir Lifschitz and Alexander Razborov. Why are there so many loop formulas? *ACM Transactions on Computational Logic*, 7(2):261–268, 2006.

[164] Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1–2):115–137, 2004.

[165] Michael L. Littman, Stephen M. Majercik, and Toniann Pitassi. Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[166] Feng Lu, Li-C. Wang, Kwang-Ting Cheng, and Ric C-Y Huang. A circuit SAT solver with signal correlation guided learning. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, pages 892–897. IEEE Computer Society, 2003.

[167] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[168] Inês Lynce and João Marques-Silva. The puzzling role of simplification in propositional satisfiability. In *Proceedings of the EPIA'01 Workshop on Constraint Satisfaction and Operational Research Techniques for Problem Solving (EPIA-CSOR'01)*, 2001.

[169] Inês Lynce and João Marques-Silva. Probing-based preprocessing techniques for propositional satisfiability. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)*, pages 105–111. IEEE Computer Society, 2003.

[170] Inês Lynce and João Marques-Silva. Efficient haplotype inference with boolean satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI'06)*, pages 104–109. AAAI Press, 2006.

[171] João Marques-Silva. Model checking with Boolean satisfiability. *Journal of Algorithms: Algorithms in Cognition, Informatics, and Logic*, 63(1–3):3–16, 2008.

[172] João Marques-Silva and Luís Guerra e Silva. Solving satisfiability in combinational circuits. *IEEE Design & Test of Computers*, 20(4):16–21, 2003.

[173] João Marques-Silva and Ines Lynce. Towards robust CNF encodings of cardinality constraints. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *LNCS*, pages 483–497. Springer, 2007.

[174] João Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[175] Fabio Massacci. Using Walk-SAT and Rel-sat for cryptographic key search. In Thomas Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 290–295. Morgan Kaufmann, 1999.

[176] Bertrand Mazure, Lakhdar Sais, and Éric Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artifificial Intelligence*, 22(3–4):319–331, 1998.

[177] David McAllester, Bart Selman, and Henry A. Kautz. Evidence for invariants in local search. In Benjamin K. Kuipers and Bonnie Webber, editors, *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*, pages 321–326. AAAI Press, 1997.

[178] Ilya Mironov and Lintao Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In Armin Biere and Carla P. Gomes, editors, *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115. Springer, 2006.

[179] David Mitchell, Bart Selman, and Hector Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465. AAAI Press, 1992.

[180] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535. ACM, 2001.

[181] Mark E.J. Newman and Cristopher Moore. Glassy dynamics in an exactly solvable spin model. *Physical Review E*, 60:5068–5072, 1999.

[182] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.

[183] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[184] Sergey I. Nikolenko. Hard satisfiable instances for DPLL-type algorithms. *Journal of Mathematical Sciences*, 126(3):1205–1209, 2005.

[185] Andreas Nonnengart, Georg Rock, and Christoph Weidenbach. On generating small clause normal forms. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the 15th International Conference on Automated Deduction (CADE'98)*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 397–411. Springer, 1998.

[186] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, USA, 1995.

[187] Duc Nghia Pham, John R. Thornton, and Abdul Sattar. Building structure into local search for SAT. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2359–2364. AAAI Press, 2007.

[188] David A. Plaisted and Steven A. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:193–304, 1986.

[189] Steven D. Prestwich and Inês Lynce. Local search for unsatisfiability. In Armin Biere and Carla P. Gomes, editors, *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06)*, volume 4121 of *Lecture Notes in Computer Science*, pages 283–296. Springer, 2006.

[190] Tuomo Pyhälä. Factoring benchmarks for SAT-solvers, 2004. `http://www.tcs.hut.fi/Software/genfacbm/`.

[191] Claude-Guy Quimper and Toby Walsh. Decomposing global grammar constraints. In Christian Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *Lecture Notes in Computer Science*, pages 590–604. Springer, 2007.

[192] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics, Second Series*, 155(1):157–187, 2002.

[193] Federico Ricci-Tersenghi, Martin Weight, and Riccardo Zecchina. Simplest random K-satisfiability problem. *Physical Review E*, 63:026702, 2001.

[194] Jussi Rintanen and Alban Grastien. Diagnosability testing with satisfiability algorithms. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 532–537. AAAI Press, 2007.

[195] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12–13):1031–1080, 2006.

[196] Irina Rish and Rina Dechter. To guess or to think? hybrid algorithms for SAT. In Eugene C. Freuder, editor, *Proceedings of the 2nd International Conference on Principles and Practice of Constraint Programming (CP'96)*, volume 1118 of *Lecture Notes in Computer Science*, pages 555–556. Springer, 1996.

[197] John A. Robinson. A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[198] Jan-Willem Roorda and Koen Claessen. SAT-based assistance in abstraction refinement for symbolic trajectory evaluation. In Thomas

Ball and Robert B. Jones, editors, *Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2006.

[199] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of constraint programming*. Series in Foundations of Artificial Intelligence. Elsevier, 2006.

[200] Jarrod A. Roy, Igor L. Markov, and Valeria Bertacco. Restoring circuit structure from SAT instances. In *Informal Proceedings of the 2004 International Workshop on Logic Synthesis*, 2004. Available at `http://www.eecs.umich.edu/~imarkov/pubs/misc/iwls04-sat2circ.pdf`.

[201] Yongshao Ruan, Henry A. Kautz, and Eric Horvitz. The backdoor key: A path to understanding problem hardness. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, pages 124–130. AAAI Press, 2004.

[202] Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004. Available at `http://www.cs.sfu.ca/~mitchell/papers/ryan-thesis.ps`.

[203] Ashish Sabharwal. SymChaff: A structure-aware satisfiability solver. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, pages 467–474. AAAI Press / The MIT Press, 2005.

[204] Sean Safarpour, Aandreas G. Veneris, Rolf Drechsler, and Joanne Lee. Managing don't cares in Boolean satisfiability. In *Proceedings of the 2004 Design, Automation and Test in Europe Conference and Exposition (DATE'04)*, pages 260–265. IEEE Computer Society, 2004.

[205] Horst Samulowitz and Fahiem Bacchus. Using SAT in QBF. In Peter van Beek, editor, *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *Lecture Notes in Computer Science*, pages 578–592. Springer, 2005.

[206] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *Online Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, 2004.

[207] Tian Sang, Paul Beame, and Henry A. Kautz. Performing bayesian inference by weighted model counting. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05)*, pages 475–482. AAAI Press / The MIT Press, 2005.

[208] Roberto Sebastiani. Applying GSAT to non-clausal formulas. *Journal of Artificial Intelligence Research*, 1:309–314, 1994.

[209] Roberto Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.

[210] Nathan Segerlind. On the relative efficiency of resolution-like proofs and ordered binary decision diagram proofs. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC'08)*, pages 100–111. IEEE Computer Society, 2008.

[211] Sakari Seitz, Mikko Alava, and Pekka Orponen. Focused local search for random 3-satisfiability. *Journal of Statistical Mechanics: Theory and Experiment*, P06006, 2005.

[212] Sakari Seitz and Pekka Orponen. An efficient local search method for random 3-satisfiability. *Electronic Notes in Discrete Mathematics*, 16:71–79, 2003.

[213] Bart Selman and Henry A. Kautz. Domain-independent extension to GSAT: Solving large structured satisfiability problems. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 290–295. Morgan Kaufmann, 1993.

[214] Bart Selman and Henry A. Kautz. An empirical study of greedy local search for satisfiability testing. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'93)*, pages 46–51. AAAI Press, 1993.

[215] Bart Selman and Henry A. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996.

[216] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343. AAAI Press, 1994.

[217] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. AMS, 1996.

[218] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446. AAAI Press, 1992.

[219] Junhao Shi, Görschwin Fey, Rolf Drechsler, Andreas Glowatz, Jürgen Schlöffel, and Friedrich Hapke. Experimental studies on SAT-based test pattern generation for industrial circuits. In *Proceedings of the 6th International Conference on ASIC, volume 2*, pages 967–970. IEEE Computer Society, 2005.

[220] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[221] Carsten Sinz and Armin Biere. Extended resolution proofs for conjoining BDDs. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *Proceedings of the 1st International Computer Science Symposium in Russia (CSR'06)*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006.

[222] Raymond M. Smullyan. *First–Order Logic*. Springer, Heidelberg, Germany, 1968.

[223] Zbigniew Stachniak and Anton Belov. Speeding-up non-clausal local search for propositional satisfiability with clause learning. In Hans Kleine Büning and Xishun Zhao, editors, *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 2008.

[224] Paul Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1167–1176, 1996.

[225] Ofer Strichman. Tuning SAT checkers for bounded model checking. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 2000.

[226] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances. In Holger H. Hoos and David G. Mitchell, editors, *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2005.

[227] Stefan Szeider. Backdoor sets for DLL subsolvers. *Journal of Automated Reasoning*, 35(1–3):73–88, 2005.

[228] Paul Tafertshofer and Andreas Ganz. SAT based ATPG using fast justification and propagation in the implication graph. In Jacob K. White and Ellen Sentovich, editors, *Proceedings of the 1999 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'99)*, pages 139–146. IEEE Computer Society, 1999.

[229] R. Michael Tanner. Explicit concentrators from generalized N-gons. *SIAM Journal on Algebraic and Discrete Methods*, 5(3):287–293, 1984.

[230] Christian Thiffault, Fahiem Bacchus, and Toby Walsh. Solving non-clausal formulas with DPLL search. In Mark Wallace, editor, *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)*, volume 3258 of *Lecture Notes in Computer Science*, pages 663–678. Springer, 2004.

[231] Ashish Tiwari, Carolyn L. Talcott, Merrill Knapp, Patrick Lincoln, and Keith Laderoute. Analyzing pathways using SAT-based approaches. In Hirokazu Anai, Katsuhisa Horimoto, and Temur Kutsia, editors, *Proceedings of the 2nd International Conference on Algebraic Biology (AB'07)*, volume 4545 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2007.

[232] Dave Tompkins and Holger H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In Holger H. Hoos and David G. Mitchell, editors, *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT'04), Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2005.

[233] Grigori S. Tseitin. On the complexity of derivation in propositional calculus. In A.O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, volume 8 of *Seminars in Mathematics, V.A. Steklov Mathematical Institute, Leningrad*, pages 115–125. Consultants Bureau, 1969. [Translated from Russian. Reprinted in J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967–1970*, pages 466–483. Springer, 1983.].

[234] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, 1987.

[235] Alasdair Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 1(4):425–467, 1995.

[236] Alasdair Urquhart. The symmetry rule in propositional logic. *Discrete Applied Mathematics*, 96–97:177–193, 1999.

[237] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[238] Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors. *Handbook of Knowledge Representation*. Elsevier, 2008.

[239] Miroslav N. Velev and Randy E. Bryant. Superscalar processor verification using efficient reductions of the logic of equality with uninterpreted functions to propositional logic. In Laurence Pierre and Thomas Kropf, editors, *Correct Hardware Design and Verification Methods, Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference (CHARME'99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 1999.

[240] Michael Wachter and Rolf Haenni. Propositional DAGs: a new graph-based language for representing boolean functions. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 277–285. AAAI Press, 2006.

[241] Toby Walsh. SAT v CSP. In Rina Dechter, editor, *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP'00)*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[242] Joost P. Warners and Hans van Maaren. A two-phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters*, 23(3-5):81–88, 1998.

[243] Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1173–1178. Morgan Kaufmann, 2003.

[244] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285. ACM, 2001.

[245] Lintao Zhang and Sharad Malik. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In Pascal Van Hentenryck, editor, *Proceedings of the 8th Principles and Practice of Constraint Programming (CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2002.

TKK DISSERTATIONS IN INFORMATION AND COMPUTER SCIENCE

TKK-ICS-D1 Lehtimäki, Pasi
Data Analysis Methods for Cellular Network Performance Optimization. 2008.

TKK-ICS-D2 Laur, Sven
Cryptographic Protocol Design. 2008.

TKK-ICS-D3 Harva, Markus
Algorithms for Approximate Bayesian Inference with Applications to Astronomical Data Analysis. 2008.

TKK-ICS-D4 Ukkonen, Antti
Algorithms for Finding Orders and Analyzing Sets of Chains. 2008.

TKK-ICS-D5 Tatti, Nikolaj
Advances in Mining Binary Data: Itemsets as Summaries. 2008.

TKK-ICS-D6 Klami, Arto
Modeling of Mutual Dependencies. 2008.

TKK-ICS-D7 Oikarinen, Emilia
Modularity in Answer Set Programs. 2008.

TKK-ICS-D8 Salojärvi, Jarkko
Inferring Relevance from Eye Movements with Wrong Models. 2008.

TKK-ICS-D9 Yang, Zhirong
Discriminative Learning with Application to Interactive Facial Image Retrieval. 2008.

ISBN 978-951-22-9641-5 (Print)
ISBN 978-951-22-9642-2 (Online)
ISSN 1797-5050 (Print)
ISSN 1797-5069 (Online)