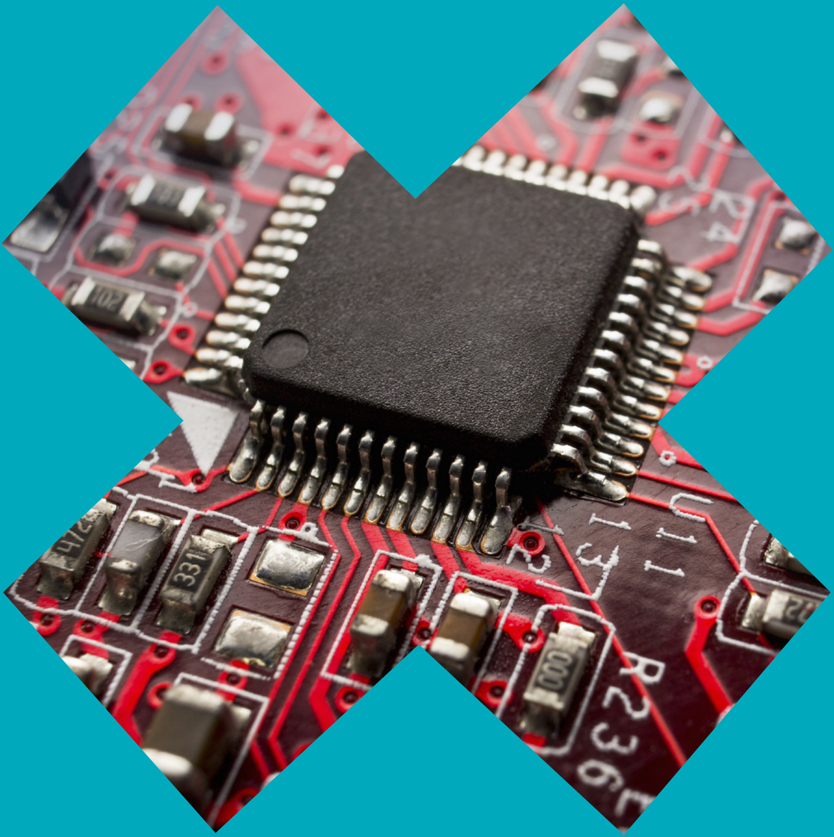
Department of Information and Computer Science

Extending SAT Solver with Parity Reasoning

Tero Laitinen



Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 177/2014

Extending SAT Solver with Parity Reasoning

Tero Laitinen

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held in the lecture hall AS2 of the school on 21 November 2014 at 12 noon.

Aalto University
School of Science
Department of Information and Computer Science

Aalto University publication series
DOCTORAL DISSERTATIONS 177/2014

© Tero Laitinen

ISBN 978-952-60-5944-0 (printed)
ISBN 978-952-60-5945-7 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-5945-7

Unigrafia Oy
Helsinki 2014

Finland

441 697
Printed matter

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author Tero Laitinen		
Name of the doctoral dissertation Extending SAT Solver with Parity Reasoning		
Publisher School of Science		
Unit Department of Information and Computer Science		
Series Aalto University publication series DOCTORAL DISSERTATIONS 177/2014		
Field of research Theoretical Computer Science		
Manuscript submitted 10 September 2014	**Date of the defence** 21 November 2014	
Permission to publish granted (date) 29 October 2014	**Language** English	
☐ **Monograph**	☒ **Article dissertation (summary + original articles)**	

Abstract

Propositional conflict-driven clause-learning (CDCL) satisfiability (SAT) solvers have been successfully applied in a number of industrial domains. In some application areas such as circuit verification, bounded model checking, logical cryptanalysis, and approximate model counting, some requirements can be succinctly captured with parity (xor) constraints. However, satisfiability solvers that typically operate in conjunctive normal form (CNF) may perform poorly with straightforward translation of parity constraints to CNF.

This work studies how CDCL SAT solvers can be enhanced to handle problems with parity constraints using the recently introduced DPLL(XOR) framework where the SAT solver is coupled with a parity constraint solver module. Different xor-deduction systems ranging from plain unit propagation through equivalence reasoning to complete incremental Gauss-Jordan elimination are presented. Techniques to analyze xor-deduction system derivations are developed, allowing one to obtain smaller clausal explanations for implied literals and also to learn new parity constraints in the conflict analysis process. It is proven that these techniques can be used to simulate a complete xor-deduction system on a restricted class of instances and allow very short unsatisfiability proofs for some formulas whose CNF translations are hard for resolution. Fast approximating tests to detect whether unit propagation or equivalence reasoning is enough to deduce all implied literals are presented. Methods to decompose sets of parity constraints into subproblems that can be handled separately are developed. The decomposition methods can greatly reduce the size of parity constraint matrices when using Gauss-Jordan elimination on dense matrices and allow one to choose appropriate xor-deduction system for each subproblem. Efficient translations to simulate equivalence reasoning and stronger parity reasoning are developed. It is shown that equivalence reasoning can be simulated by adding a polynomial amount of redundant parity constraints to the problem, but without using additional variables, an exponential number of parity constraints are needed in the worst case. It is proven that resolution simulates equivalence reasoning efficiently. The presented techniques are experimentally evaluated on a variety of challenging problems originating from a number of encryption ciphers and from SAT Competition benchmark instances.

Keywords propositional satisfiability, parity reasoning		
ISBN (printed) 978-952-60-5944-0	**ISBN (pdf)** 978-952-60-5945-7	
ISSN-L 1799-4934	**ISSN (printed)** 1799-4934	**ISSN (pdf)** 1799-4942
Location of publisher Helsinki	**Location of printing** Helsinki	**Year** 2014
Pages 225	**urn** http://urn.fi/URN:ISBN:978-952-60-5945-7	

Aalto-yliopisto

Tiivistelmä

Aalto-yliopisto, PL 11000, 00076 Aalto www.aalto.fi

Tekijä Tero Laitinen	
Väitöskirjan nimi Lauselogiikan toteutuvuustarkistimen laajentaminen pariteettipäättelyllä	
Julkaisija Perustieteiden korkeakoulu	
Yksikkö Tietojenkäsittelytieteen laitos	
Sarja Aalto University publication series DOCTORAL DISSERTATIONS 177/2014	
Tutkimusala Tietojenkäsittelyteoria	
Käsikirjoituksen pvm 10.09.2014	**Väitöspäivä** 21.11.2014
Julkaisuluvan myöntämispäivä 29.10.2014	**Kieli** Englanti
☐ **Monografia**	☒ **Yhdistelmäväitöskirja (yhteenveto-osa + erillisartikkelit)**

Tiivistelmä

Hakukonflikteista oppivia lauselogiikan toteutuvuustarkastimia on menestyksekkäästi sovellettu ongelmanratkaisuun useissa käytännön sovellutuksissa. Tietyillä sovellusalueilla, kuten piiriverifiointi, rajoitettu mallintarkastus, looginen kryptoanalyysi ja approksimoiva ratkaisumallien lukumäärän laskenta, ongelmakuvauksien vaatimuksia voidaan ilmaisuvoimaisesti mallintaa pariteettirajoitteilla. Ongelmat, jotka sisältävät pariteetti- eli xor-rajoitteita, voivat tosin olla erityisen vaativia toteutuvuustarkistimille, jotka käsittelevät ongelmaa konjunktiivisessa normaalimuodossa. Tässä väitöskirjassa tutkitaan, miten hakukonflikteista oppivia toteutuvuustarkistimia voidaan kehittää käsittelemään pariteettirajoitteita sisältäviä ongelmia käyttäen aikaisemmin julkaistua DPLL(XOR)-hakumenetelmäkehystä, jossa toteutuvuustarkistimeen yhdistetään pariteettirajoitteita käsittelevä ratkaisinmoduuli. Työssä esitellään erilaisia pariteettipäättelyjärjestelmiä kuten yksikköpropagaatio, ekvivalenssipäättely ja täydellisen pariteettipäättelyn tuottava inkrementaalinen Gauss-Jordan-eliminaatio. Pariteettirajoitejohtoihin perustuvien pariteettipäättelyjärjestelmien analysoimiseksi esitellään tekniikoita, joilla voidaan johtaa lyhyempiä klausuulipohjaisia selityksiä implikoiduille literaaleille ja joita voidaan käyttää uusien pariteettirajoitteiden oppimiseksi hakukonfliktien käsittelyn yhteydessä. Työssä osoitetaan, että näillä tekniikoilla voidaan simuloida täydellistä pariteettipäättelyjärjestelmää rajatussa ongelmajoukossa ja niiden avulla voidaan tuottaa lyhyitä ongelman toteutumattomuuden osoittavia todistuksia eräille ongelmille, joiden kuvaukset konjunktiivisessa normaalimuodossa ovat vaikeita resoluutiolle. Väitöskirjassa käsitellään approksimoivia luokittelumenetelmiä, joilla voidaan päätellä, että annetulle ongelmalle riittää käyttää yksikköpropagaatiota tai ekvivalenssipäättelyä kaikkien implikoitujen literaalien päättelyyn. Työssä näytetään, miten erilaisia menetelmiä osittaa ongelma erillisiksi aliongelmiksi voidaan soveltaa pariteettirajoitejoukkoihin. Ositusmenetelmiä voidaan käyttää pienentämään Gauss-Jordan eliminaatiossa käytettävien matriisien kokoa, kun käytetään tiivistä matriisiesitystä, ja jokaiselle ositetulle aliongelmalle voidaan valita sopiva pariteettipäättelyjärjestelmä. Pariteettipäättelyn simulointia tutkitaan käyttäen ekvivalenssipäättelyä ja vahvempaa pariteettipäättelyä simuloivia käännöksiä, jotka mahdollistavat olemassaolevien toteutuvuustarkistimien hyödyntämisen. Työssä osoitetaan, että ekvivalenssipäättelyä voidaan simuloida lisäämällä polynominen määrä ylimääräisiä pariteettirajoitteita, mutta ilman lisämuuttujia tarvitaan eksponentiaalinen määrä pariteettirajoitteita pahimmassa tapauksessa. Työssä näytetään, että resoluutio simuloi ekvivalenssipäättelyä tehokkaasti. Esiteltyjä tekniikoita arvioidaan kokeellisesti monilla haastavilla ongelmilla, jotka on tuotettu salausmenetelmistä ja SAT Competition-kilpailuongelmista.

Avainsanat lauselogiikan toteutuvuusongelma, pariteettipäättely		
ISBN (painettu) 978-952-60-5944-0	**ISBN (pdf)** 978-952-60-5945-7	
ISSN-L 1799-4934	**ISSN (painettu)** 1799-4934	**ISSN (pdf)** 1799-4942
Julkaisupaikka Helsinki	**Painopaikka** Helsinki	**Vuosi** 2014
Sivumäärä 225	**urn** http://urn.fi/URN:ISBN:978-952-60-5945-7	

Contents

Preface

This thesis reports the results of the fruitful collaboration between me, docent Tommi Junttila, and professor Ilkka Niemelä in the Computational Logic research group.

My research has been funded jointly by Aalto University, Helsinki Doctoral Programme in Computer Science, and Finnish Centre of Excellence in Computational Inference Research (COIN).

I wish to thank Prof. Ilkka Niemelä and Docent Tommi Junttila for their insightful and patient guidance. Their long-term commitment to research on theoretical computer science is hopefully reflected in this thesis.

I want to express my gratitude towards my colleagues at the Department of Information and Computer Science for laughters that have released the tension of sweating on research problems.

I am grateful to my friends for supporting me and sharing rich experiences outside work as well.

I feel a deep sense of gratitude to my parents and brother who have provided a world of opportunities and have always been there for me.

In the preface of my master's thesis, I thanked Oana for all the sunshine making the future look brighter. I no longer need to wait for the bright future to happen. I thank my wife, Oana, and our two sons Christian and Alex for every moment spent together.

Helsinki, October 31, 2014,

Tero Laitinen

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Tero Laitinen and Tommi Junttila and Ilkka Niemelä. Equivalence Class Based Parity Reasoning in DPLL(XOR). In *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011*, Boca Raton, FL, USA, November 7-9, 2011, 649-658, November 2011.

II Tero Laitinen and Tommi Junttila and Ilkka Niemelä. Conflict-Driven XOR-Clause Learning. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference*, Trento, Italy, June 17-20, 2012, 383-396, June 2012.

III Tero Laitinen and Tommi Junttila and Ilkka Niemelä. Classifying and Propagating Parity Constraints. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012*, Québec City, QC, Canada, October 8-12, 2012, 357-372, October 2012.

IV Tero Laitinen and Tommi Junttila and Ilkka Niemelä. Extending Clause Learning SAT Solvers with Complete Parity Reasoning. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012*, Athens, Greece, November 7-9, 2012, 65-72, November 2012.

V Tero Laitinen and Tommi Junttila and Ilkka Niemelä. Simulating Parity Reasoning. In *19th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, Stellenbosch, South Africa, December 15-19, 2013, 568-583, December 2013.

Author's Contribution

Publication I: "Equivalence Class Based Parity Reasoning in DPLL(XOR)"

The author is responsible for development and description of the EC deduction system and associated additional techniques, and conducting and reporting the experiments. The proof for showing the partial equivalence of SUBST and EC deduction systems and formal description of the EC implementation is due to Docent Junttila. The ideas in the DPLL(XOR) framework are by Prof. Niemelä and Docent Junttila.

Publication II: "Conflict-Driven XOR-Clause Learning"

The author is responsible for development and description of the method of conflict-driven XOR-clause learning, showing that resolution cannot polynomially simulate parity explanations, and conducting and documenting the experiments. The idea and the description of parity explanations is due to Docent Junttila.

Publication III: "Classifying and Propagating Parity Constraints"

The author is responsible for description of the connection between xor-cycles and equivalence reasoning, the method for the approximating test to detect whether equivalence reasoning is enough to detect all implied literals, the descriptions of all translations to simulate equivalence reasoning with unit propagation, and conducting and reporting the experiments related to equivalence reasoning and instance classification. Development and description of the method of clausification of tree-like parts, conducting and reporting the related experiments is due to Docent Junttila.

Publication IV: "Extending Clause Learning SAT Solvers with Complete Parity Reasoning"

The author is responsible for the description of the method for exploiting biconnected components when using dense matrix representation with Gauss-Jordan elimination, documenting the implementation details of the incremental Gauss-Jordan solver module, the description of the method for eliminating xor-internal variables, and conducting and reporting the experiments. The formal description of incremental Gauss-Jordan module and all proofs are due to Docent Junttila.

Publication V: "Simulating Parity Reasoning"

The author is responsible for the idea that parity explanations simulate Gauss-Jordan elimination on nondeterministic unit propagation derivations on a restricted class of instances, describing the method of simulating stronger parity reasoning with unit propagation and associated proof, describing the method of propagation-preserving xor-simplification and associated proof, and conducting and reporting the experiments. The proofs showing resolution polynomially simulates equivalence reasoning, and parity explanations simulate Gauss-Jordan elimination on nondeterministic unit propagation derivations on a restricted class of instances are due to Docent Junttila.

Contributions of the Publications

I This work develops a new xor-deduction system in the DPLL(XOR) framework. The new xor-deduction system EC uses an alternative approach to implement equivalence reasoning by tracking equivalence classes of variables whereas the earlier xor-deduction system SUBST uses substitution to implement equally powerful equivalence reasoning.

II This work develops parity explanations, a method to possibly obtain shorter conflict clauses compared to earlier implicative explanations when solving instances with parity constraints. A new simple xor-deduction system UP that performs plain unit propagation on xor-constraints is introduced. The work shows that by choosing assumed literals appropriately, parity explanations can be used to refute unsatisfiable formulas that do not have polynomial-size resolution refutations. The work proposes a method to selectively add parity explanations as learned xor-constraints during the search.

III This work develops two fast approximating tests for deciding whether unit propagation or equivalence reasoning is enough to achieve full propagation in a given parity constraint test. The work also develops three translations for simulating equivalence reasoning with unit propagation, and shows that without additional variables, such simulation is exponential in the worst case.

IV This work develops a new xor-deduction system based on incremental Gauss-Jordan elimination. The new xor-deduction systems allows SAT solver to be extended with complete parity reasoning. The work develops a method to decompose parity constraint conjunctions by exploiting biconnected components. The decomposition method allows smaller memory usage and faster propagation for incremental Gauss-Jordan elimination when using dense matrix representation. The work shows how some variables occurring only in the xor-part can be eliminated while preserving the biconnected component decomposition.

V This work studies how stronger parity reasoning techniques in the DPLL(XOR) framework can be simulated by simpler systems: resolution, unit propagation, and parity explanations. It is shown that resolution polynomially simulates equivalence reasoning. It is proven that parity explanations simulate Gauss-Jordan elimination on nondeterministic unit propagations on a restricted class of instances. The work develops a new translation that allows unit propagation to effectively simulate stronger parity reasoning ranging from equivalence reasoning to complete parity reasoning.

1. Introduction

This thesis develops methods to solve the propositional satisfiability (SAT) problem which is to assign a truth value, true or false, to each variable in a Boolean formula in such a way the formula evaluates to true. This work focuses on SAT problems that involve parity (xor) constraints. As a method to solve such problems, this work builds upon modern SAT solvers capable of conflict-driven clause-learning (CDCL).

Conflict-driven clause-learning SAT solvers have been successfully applied in a number of industrial application domains including AI planning, model checking of software and hardware systems, package management in software distributions (see e.g. Silva *et al.* [2009]). In these solvers, an instance of the SAT problem is typically represented in conjunctive normal form (CNF) which allows very efficient algorithms to explore large search spaces. However, these algorithms do not typically scale well for problems involving parity constraints that are commonly used in application domains such as circuit verification, bounded model checking, logical cryptanalysis, and approximate model counting.

This thesis has a number of practical and theoretical results that have three interesting implications. First, the application domains where parity constraints are used as a part of the modeling language benefit directly from the pragmatic results enabling SAT solvers to solve even larger problems. Second, the theoretical results provide a foundation for building next generation SAT solvers capable of handling parity constraints effectively. Third, the use of parity constraints has been avoided in SAT problem modeling because of its reputation for hindering the performance of SAT solvers. The results in this thesis encourage to use parity constraints as a part of the modeling language, because parity constraints provide structure in the problem description and that structure can be exploited when solving the problem. The importance of the results in this thesis can then extend to yet unknown important application domains.

1.1 SAT Problem and CDCL SAT Solvers

The propositional satisfiability (SAT) problem is one of the NP-complete problems meaning that, according to the current understanding of computational complexity theory, solving the SAT problem is intractable in the sense that solving an instance of the SAT problem may require an exponential amount of time with respect to the size of the problem instance. Despite this dreadful worst-case behavior, many important real-world problems can be translated to and solved efficiently as SAT problems thanks to significant progress in SAT solving technology since 1990s and especially to the considerable software engineering effort put into modern conflict-driven clause-learning CDCL SAT solvers. The SAT problem is an important restricted class of CSPs (Constraint Satisfaction Problems) that typically involve non-Boolean variables and may mix different constraint types. In constraint programming, customized search strategies may be defined when solving CSPs whereas SAT solvers typically rely on heuristic built-in search strategies. Dechter [2003] gives a comprehensive overview of the theory of CSPs.

The basis for modern CDCL SAT solvers lies in the DPLL algorithm by Davis *et al.* [1962]. The DPLL algorithm performs a complete backtracking search on an instance of the SAT problem represented in conjunctive normal form (CNF). The search alternates between branching and propagation. In a branching step, a variable and a truth value for it are selected, and the formula is simplified accordingly. As a result, in the propagation step, it may be possible to deduce truth values of some other variables. If it is deduced that a part of the formula cannot evaluate to true under the current truth assignment, then the DPLL algorithm backtracks, that is, restores the state of the search to a previously unexplored branch of the search tree.

Since the inception of the DPLL algorithm, a number of improvements have been proposed to it ultimately leading to the current CDCL SAT solvers capable of handling formulas with millions of variables. When the basic DPLL algorithm backtracks, it does not store information about the *conflicting* truth assignment that causes a part of the formula evaluate to false and the search to backtrack. A method to store information about such conflicts by adding *learned clauses* to the formula being solved was proposed by Silva and Sakallah [1996]. To avoid the SAT solver from getting stuck in a single region of the search space, a method to randomize the search and restart the search periodically from the beginning was presented by Gomes *et al.* [1998]. To implement the propagation step efficiently for larger formulas, efficient lazy data structures and compatible light-weight variable selection heuristics were developed by Moskewicz *et al.* [2001] and further elaborated by Ryan [2004] and Lewis *et al.* [2005]. To prevent learned clauses from consuming too much memory and slowing down propagation excessively, deletion policies for selectively discarding some learned clauses were proposed by Goldberg and Novikov [2007]. A more detailed overview of CDCL SAT solving techniques can be found in Silva *et al.* [2009].

The (CDCL) SAT solving technology continues to develop at a fast pace, so it can be challenging to understand which are the most important techniques governing the performance of a SAT solver. The solver minisat by Eén and Sörensson [2003] provides a clean and extensible implementation of a conservative number of well-considered SAT solving techniques. Besides being the basis for many award-winning SAT solvers in the international SAT competition (www.satcompetition.org), it is commonly used as a reference point for benchmarking new SAT solving techniques, and so it is used in this thesis.

1.2 Satisfiability Modulo Theories and DPLL(T) Framework

Encoding an application-level problem as a Boolean formula can be prohibitively cumbersome for two reasons: (i) it can be tedious to capture the problem in the propositional domain, and (ii) solving the problem encoded as a Boolean formula can be inefficient. Satisfiability Modulo Theories (SMT) is an attempt tackle these challenges. Satisfiability Modulo Theories can be seen as a generalization of the propositional satisfiability problem in a sense that it relaxes the interpretation of propositional statements in the Boolean formula. In the SMT problem, instead of being plain Boolean variables, the propositional statements are formulas of some background theory T, a decidable quantifier-free fragment of first-order logic. Such background theories may, for example, consider equality with uninterpreted functions (Nelson and Oppen [1980]), linear integer arithmetic (Dutertre and de Moura [2006]), or fixed-size bit vectors (Cyrluk *et al.* [1997]). We shortly summarize the two successful approaches, "eager" and "lazy", to solving SMT and introduce the DPLL(T) framework that has inspired the development of the DPLL(XOR) framework introduced in the next chapter. More complete overviews on Satisfiability Modulo Theories can be found in Sebastiani [2007] and Barrett *et al.* [2009].

In the eager approach, an SMT instance is translated to an equisatisfiable Boolean formula by expanding the domains of the background theory variables and incorporating enough consequences of the background theory. The resulting Boolean formula may be substantially larger than the original SMT instance. A benefit of using the eager approach is that an unmodified SAT solver can be used to solve the translated SMT instance.

In the lazy approach, a SAT solver is extended with a solver module that takes care of background theory-specific inference. The solver module can then use specialized data structures and algorithms, which can be much more efficient than translating the SMT instance to a Boolean formula.

The DPLL(T) approach to Satisfiability Modulo Theories by Nieuwenhuis *et al.* [2006] defines a formal model of CDCL SAT solver and a minimal interface to integrate a solver module for a background theory T. Such a solver module needs only to deal with conjunctions of theory literals, tell the SAT solver which theory literals become implied, and provide explanations for implied theory literals when required by the SAT solver's conflict analysis.

1.3 Satisfiability and Parity Constraints

The use of DPLL-based algorithms requires that the problem be represented in conjunctive normal form (CNF). Few problems have natural representations in conjunctive normal form, so SAT problems are usually converted to CNF from an intermediate representation, e.g. using the translation by Tseitin [1968]. Straightforward Tseitin-translation of a problem instance to CNF may result in poor performance, especially in the case of parity (xor) constraints. Parity constraints (linear equations modulo two) can be efficiently solved by Gaussian elimination in polynomial time. However, parity constraints, when translated to CNF, have been shown by Urquhart [1987] to be very difficult for *resolution*, a refutation-based theorem proving technique for propositional formulas, which is equivalent to the underlying proof system in CDCL SAT solvers (Pipatsrisawat and Darwiche [2011]). Due to this inherent hardness of parity constraints, several approaches to combining CNF-level and parity reasoning have been proposed. Baumgartner and Massacci [2000] develop a formal decision procedure to solve formulas consisting of CNF and parity constraints. The solvers EqSatz by Li [2000b], 2cls+eq by Bacchus [2002], lsat by Ostrowski *et al.* [2002], march_eq by Heule *et al.* [2004], MoRSat by Chen [2009], cryptominisat by Soos [2010], and the solver by Han and Jiang [2012] implement a variety of important techniques to integrate parity reasoning in a SAT solver. Katsirelos and Simon [2012] consider unifying the reasoning on CNF and parity constraints using Polynomial Calculus with Resolution (Alekhnovich *et al.* [2002]). Weaver [2012] uses state machines to represent constraints, including parity constraints, in a SAT solver. Gwynne and Kullmann [2014] consider the problem of finding good CNF-representations for parity constraints. A more detailed comparison against this considerable amount of related work will be made in the following chapters where relevant.

1.4 Structure of the thesis

The results in this thesis are organized as follows.

Chapter 2 introduces formal notation, the DPLL(XOR) framework presented in Laitinen *et al.* [2010] and lists the benchmark families used for experimental evaluation.

Chapter 3 presents and experimentally evaluates four different xor-deduction systems.

Chapter 4 develops and experimentally evaluates techniques to give better explanations for literals deduced with xor-deduction systems.

Chapter 5 explores the structure of our benchmark instances and develops efficient approximating tests to decide whether unit propagation or equivalence reasoning is enough to achieve full propagation in a given set of parity constraints.

Chapter 6 develops and experimentally evaluates techniques to improve solving performance by decomposing the parity constraints into parity constraint clusters that can be handled separately.

Chapter 7 studies to what extent simpler parity reasoning systems can simulate stronger parity reasoning systems in the DPLL(XOR) framework, develops and experimentally evaluates translations to simulate parity reasoning on existing SAT solvers, and considers how some propagation engines and proof systems relate to each other on a more fundamental level.

Chapter 8 concludes the work and discusses some open questions for future work.

2. Preliminaries

We are interested in solving propositional formulas consisting of a CNF-part ϕ_{or} and an xor-part ϕ_{xor}. Starting with some formal notation, we now introduce the DPLL(XOR) framework presented in Laitinen *et al.* [2010] to extend a CDCL SAT solver with an xor-reasoning module. The article Laitinen *et al.* [2010] documents the main results of the first author's Master's thesis, so it is not included in this thesis.

2.1 Parity constraint formulas

Let $\mathbb{B} = \{\bot, \top\}$ be the set of truth values "false" and "true". A literal is a Boolean variable x or its negation $\neg x$ (as usual, $\neg\neg x$ will mean x), and a clause is a disjunction of literals. If ϕ is any kind of formula or equation, (i) $\mathrm{vars}(\phi)$ is the set of variables occurring in it, (ii) $\mathrm{lits}(\phi) = \{x, \neg x \mid x \in \mathrm{vars}(\phi)\}$ is the set of literals over $\mathrm{vars}(\phi)$, and (iii) a truth assignment for ϕ is a, possibly partial, function $\tau : \mathrm{vars}(\phi) \to \mathbb{B}$. A truth assignment satisfies (i) a variable x if $\tau(x) = \top$, (ii) a literal $\neg x$ if $\tau(x) = \bot$, and (iii) a clause $(l_1 \vee .. \vee l_k)$ if it satisfies at least one literal l_i in the clause.

An *xor-constraint* is an equation of the form $x_1 \oplus ... \oplus x_k \equiv p$, where the x_is are Boolean variables and $p \in \mathbb{B}$ is the parity. We implicitly assume that each xor-constraint is in a *normal form* such that duplicate variables are always removed from the equations, e.g. $x_1 \oplus x_2 \oplus x_1 \oplus x_3 \equiv \top$ is always simplified into $x_2 \oplus x_3 \equiv \top$. If the left hand side does not have variables, then it equals to $\bot$; the equation $\bot \equiv \top$ is a contradiction and $\bot \equiv \bot$ a tautology. We identify the xor-constraint $x \equiv \top$ with the literal x, $x \equiv \bot$ with $\neg x$, $\bot \equiv \bot$ with $\top$, and $\top \equiv \bot$ with $\bot$. A truth assignment τ satisfies an xor-constraint $(x_1 \oplus ... \oplus x_k \equiv p)$ if $\tau(x_i)$ is defined for all $x_i \in \{x_1, \dots, x_k\}$ and the sequence $\langle \tau(x_1), \dots, \tau(x_k) \rangle$ contains an odd (even) number of occurrences of the truth value $\top$ if p is $\top$ ($\bot$). We use

$D[x/Y]$ to denote the xor-constraint obtained from D by substituting the variable x in it with Y. For instance, $(x_1 \oplus x_2 \oplus x_3 \equiv \top)[x_1/x_2 \oplus \top] = x_2 \oplus \top \oplus x_2 \oplus x_3 \equiv \top = x_3 \equiv \bot$. The straightforward CNF translation of an xor-constraint D is denoted by $\mathrm{cnf}(D)$; for instance, $\mathrm{cnf}(x_1 \oplus x_2 \oplus x_3 \equiv \bot) = (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$. We define the linear combination of two xor-constraints, $D = (x_1 \oplus ... \oplus x_k \equiv p)$ and $E = (y_1 \oplus ... \oplus y_l \equiv q)$, by $D + E = (x_1 \oplus ... \oplus x_k \oplus y_1 \oplus ... \oplus y_l \equiv p \oplus q)$. An xor-constraint $E = (x_1 \oplus ... \oplus x_k \equiv p)$ with $k \geq 1$ is a *prime implicate* of a satisfiable xor-constraint conjunction ϕ_{xor} if (i) $\phi_{\text{xor}} \models E$ but (ii) $\phi_{\text{xor}} \not\models E'$ for all xor-constraints E' for which $\mathrm{vars}(E')$ is a proper subset of $\mathrm{vars}(E)$.

A *CNF-xor formula* is a conjunction $\phi_{\text{or}} \wedge \phi_{\text{xor}}$, where ϕ_{or} is a conjunction of clauses and ϕ_{xor} is a conjunction of xor-constraints. A truth assignment satisfies $\phi_{\text{or}} \wedge \phi_{\text{xor}}$ if it satisfies every clause and xor-constraint in it.

2.2 Fundamental Properties of Linear Combinations

Some fundamental, easy to verify properties are $D + D + E = E$, $D \wedge E \models D + E$, $D \wedge E \models D \wedge (D + E)$, and $D \wedge (D + E) \models D \wedge E$.

The logical consequence xor-constraints of an xor-constraint conjunction ψ are exactly those that are linear combinations of the xor-constraints in ψ:

Lemma 1 (Lem. 5 of Laitinen *et al.* [2012]). *Let ψ be a conjunction of xor-constraints. Now ψ is unsatisfiable if and only if there is a subset S of xor-constraints in ψ such that $\sum_{D \in S} D = (\bot \equiv \top)$. If ψ is satisfiable and E is an xor-constraint, then $\psi \models E$ if and only if there is a subset S of xor-constraints in ψ such that $\sum_{D \in S} D = E$.*

2.3 Unit propagation

Given a clause $C = l_1 \vee \cdots \vee l_n$, and τ be a partial truth assignment such that i) τ is defined for $|\mathrm{vars}(C)| - 1$ variables in $\mathrm{vars}(C)$, and ii) τ does not satisfy the clause C, *unit propagation* is the deduction step of inferring an augmented truth assignment τ' otherwise identical to τ but i) it is defined for all variables in $\mathrm{vars}(C)$, and ii) it satisfies the clause C. For instance, given the clause $C = (a \vee b \vee c)$ and the partial truth assignment τ such that $\tau(a) = \bot$ and $\tau(b) = \bot$, the partial truth assignment τ' fulfilling the conditions is obtained by setting $\tau'(a) = \tau(a) = \bot$, $\tau'(b) = \tau(b) = \bot$, and

$\tau'(c) = \top$.

2.4 DPLL(XOR)

As in the DPLL(T) approach to solving Satisfiability Modulo Theories, the DPLL(XOR) framework consists of a CDCL SAT solver and an *xor-reasoning module*. The search on a CNF-xor formula $\phi_{\text{or}} \wedge \phi_{\text{xor}}$ is driven by the CDCL SAT solver on the CNF-part ϕ_{or} employing appropriate variable selection, branching, conflict analysis, backtracking, and restarting facilities. While the CDCL SAT solver tries to extend an initially empty truth assignment over the variables of $\phi_{\text{or}} \wedge \phi_{\text{xor}}$ to a satisfying truth assignment, it uses the xor-reasoning module (i) to perform (possibly incomplete) checks to deduce whether the current partial truth assignment can still be extended to satisfy the xor-part ϕ_{xor}, and (ii) to extend the current partial truth assignment by deduction on the xor-part ϕ_{xor}.

Figure 2.1 shows the essential skeleton of the DPLL(XOR) search procedure. When the search procedure starts in lines 1-2, the xor-reasoning module is initialized with the xor-part ϕ_{xor} and the truth assignment is initially empty. The loop in lines 3-19 is run until a solution is found or it has been proven that no solution exists. In line 4, the CDCL SAT applies basic unit propagation and can infer values on some variables extending the truth assignment. The unit propagation in line 4 can also cause a *conflict* meaning that the current truth assignment cannot be extended to a solution for the instance. If unit propagation succeeds without causing a conflict (line 5), then the newly deduced literals by CNF-part unit propagation are communicated to the xor-reasoning module as *xor-assumptions* in line 6. The xor-reasoning module is then requested to run its propagation algorithms[1] and it may return one or more *xor-implied literals* in line 7 If $l_1, \ldots, l_k$ are the xor-assumptions communicated to the xor-reasoning module by ASSIGN method and $\hat{l}$ is an xor-implied literal returned by DEDUCE method, then $\hat{l}$ is a logical consequence of the xor-part and the xor-assumptions, i.e. $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k \models \hat{l}$ holds. The CDCL SAT solver requires that a literal in the current truth assignment have an *implying clause*, that is, a clause that forces the value of the literal by unit propagation on the values of literals appearing earlier in the truth assignment,

[1]It is important that the xor-reasoning module carry out the deduction on the xor-part incrementally to avoid computing the same intermediate results many times.

which in a typical implementation is a sequence of literals instead of a set. The xor-reasoning module is requested with the EXPLAIN method to supply an implying clause for each of the new xor-implied literals in line 9[2]. If the xor-reasoning module deduces that the xor-part simplified with the xor-assumptions does not have a solution (*xor-conflict*) or an xor-implied literal causes a conflict in the CNF-part, the implying clause for the xor-implied literal in line 10 is then used as a conflict clause for the conflict analysis. Otherwise, each new xor-implied literal is added to the current truth assignment in line 11. The condition in line 12 makes sure propagation is saturated in both CNF-part and xor-part before branching the search with a heuristically selected unassigned literal (line 18) or returning SAT result (line 19). The propagation steps between branching steps are said to belong to the same *decision level*. If a conflict occurs either in the CNF-part or in the xor-part, a normal conflict analysis procedure and associated backtracking (or returning UNSAT result) is performed in lines 14-16. The internal state of the xor-reasoning module is restored to a previous state consistent with the CNF-part solver's state after backtracking.

It is observed by Katsirelos and Simon [2012] that restricting the communication between the CNF-part and the xor-part in DPLL(XOR) to xor-assumptions and xor-implied literals hinders the strength of the overall proof system. In their theoretical study, the CNF-xor formula is represented as a conjunction of polynomials to unify the reasoning on the CNF-part and the xor-part. However, it is not clear to us whether such a system can be implemented efficiently.

The Gaussian elimination subroutine in the solver cryptominisat by Soos [2010] and the Gauss-Jordan elimination subroutine in the solver by Han and Jiang [2012] can be seen as xor-reasoning modules.

2.5 Other related work

CDCL SAT solvers and the pseudo-Boolean problem. CDCL SAT solvers have been used as a top-level search engine to solve the *pseudo-Boolean problem* which can be seen as a similar combined satisfiability problem where CNF is extended with another constraint type over Boolean vari-

[2] An actual implementation may postpone computing implying clauses and produce them on demand only for those xor-implied literals that are used to deduce a conflict in the CNF-part.

```
1.  initialize xor-module $M$ with $\phi_{\text{xor}}$
2.  $\tau = \langle\rangle$      /*sequence of literals (the truth assignment)*/
3.  while true:
4.      $(\tau', \mathit{confl}) = \text{UNITPROP}(\phi_{\text{or}}, \tau)$    /*standard unit propagation*/
5.      if not $\mathit{confl}$:      /*apply xor-reasoning*/
6.          for each literal $l$ in $\tau'$ but not in $\tau$: $M.\text{ASSIGN}(l)$
7.          $(\hat{l}_1, ..., \hat{l}_k) = M.\text{DEDUCE}()$
8.          for $i = 1$ to $k$:
9.              let $C = M.\text{EXPLAIN}(\hat{l}_i)$
10.             if $\hat{l}_i = \bot$ or $\neg\hat{l}_i$ in $\tau'$: $\mathit{confl} = C$, break
11.             else if $\hat{l}_i \notin \tau$: add $\hat{l}_i^C$ to $\tau'$
12.         if $k > 0$ and not $\mathit{confl}$: continue    /*unit propagate further*/
13.     let $\tau = \tau'$
14.     if $\mathit{confl}$:      /*standard Boolean conflict analysis*/
15.         analyze conflict, learn a conflict clause
16.         backjump or return UNSAT if not possible
17.     else:
18.         assign a heuristically selected unassigned literal in $\phi_{\text{or}}$ to $\tau$
19.         or return SAT if no such variable exists
```

Figure 2.1. The essential skeleton of DPLL(XOR)

ables. Formally, a linear pseudo-Boolean constraint $a_1 l_1 + \cdots + a_n l_n \geq b$ is an inequality on the weighted sum of Boolean literals where $a_i, b \in \mathbb{Z}_{\geq 0}$ and literals are interpreted as integers, e.g. given a truth assignment τ, if $\tau(l_i) = \top$, then l_1 evaluates to 1 in the pseudo-Boolean constraint. When all the coefficients a_i are equal to 1, then the linear pseudo-Boolean constraint is referred to as a *cardinality constraint*. If also the constant b is equal to 1, then the linear pseudo-Boolean constraint is equivalent to a regular clause. Problems consisting of pseudo-Boolean constraints may also involve optimization with respect to a linear objective function, and then correspond to *0-1 integer linear programming* (ILP) problems. Manquinho and Roussel [2006] document the first evaluation of pseudo-Boolean solvers which was organized as a subtrack of the SAT Competition in 2005. An important implementation detail, which does not arise with parity constraints, is the issue with big integer coefficients that may cause overflows before or during the search if not handled properly. Eén and Sörensson [2006] describe how linear pseudo-Boolean constraints can be translated to regular clauses that can be handled by a standard SAT solver. Their solver MiniSat+ implements these techniques and performs comparably to solvers that natively support pseudo-Boolean constraints. A more recent solver library Sat4j by Berre and Parrain [2010] supports pseudo-Boolean constraints and, while it is not competitive with the latest SAT solvers, its modular design allows easy adoption in software systems.

Parity reasoning in look-ahead solvers. First parity reasoning techniques have been developed in *look-ahead* solvers that do not perform conflict-driven clause learning. A look-ahead solver typically employs a search tree where the formula is simplified with more assumptions at each level of the search tree. The look-ahead part in the solver implies the use of a heuristic function to i) prune the search tree by testing possible assumptions before choosing one, and ii) guide the search to a more promising "direction". The solver EqSatz by Li [2000b], march_eq by Heule *et al.* [2004], and 2cls+eq by Bacchus [2002] successfully combine look-ahead and parity reasoning techniques. The inference rules (or related parity reasoning techniques) used in the solvers EqSatz, march_eq and 2cls+eq are compared to those of the xor-deduction system Subst in Section 3.2. The preprocessing step involving Gaussian Elimination used in the solver march_eq is described in more detail in Section 3.4 relating it to the xor-deduction system IGJ. The solver MoRsat by Chen [2009] combines look-ahead techniques, parity reasoning, and conflict-driven clause learning. The use of watched literals in the solver MoRsat is related to the xor-deduction system UP in Section 3.1.

State-based approach to parity reasoning. Weaver [2012] uses state machines called SMURFs to represent constraints to allow efficient propagation and conflict detection during search. A SMURF is an acyclic Mealy machine (Mealy [1955]) defined by a 6-tuple $(S, S_0, \Sigma, \Lambda, T, G)$ where S is a finite set of states, $S_0 \in S$ is the initial state, Σ is a finite input alphabet set, Λ is a finite output alphabet set, $T : S \times \Sigma \to S$ is a transition function, and $G : S \times \Sigma \to \Lambda$ is an output function. A state in a SMURF represents a function and transitions represents partial assignments to variables of the function. A SMURF state representing a Boolean function f with n variables has $2n$ transitions, each labeled by a literal representing one of the possible assignments to f, and at most 3^n states. The output function G can be used to encode useful information such as inference and heuristic computations, e.g. xor-implied literals. A SMURF can be constructed directly from a Binary Decision Diagram (BDD, Akers [1978]) and then it can be compressed by i) exploiting symmetries in typical functions, e.g. $\vee$, $\wedge$, and $\oplus$, and ii) introducing "counter states". An attractive property of SMURFs is that arbitrary Boolean functions can be represented and complete inference information can be encoded in transitions. However, constructing SMURFs may be computationally prohibitive. Weaver [2012] shows how to factor xor-constraints from Boolean functions to create special SMURF transitions that infer xor-constraints during search. These inferred xor-constraints are then passed to a Gaussian elimination solver used in the solver SBSAT. Constructing a SMURF to improve propagation performance and proof system strength prior to search bears a resemblance to our EC- and GE-simulation formulas presented in Chapter 7 where the idea is to add redundant xor-constraints to the formula to enable unit propagation to deduce more xor-implied literals.

Parity reasoning as a preprocessing step. The solver lsat by Ostrowski *et al.* [2002] extracts equations of the form $y = x_1 \Box \ldots \Box x_n$, where y is the *output variable*, $x_1, \ldots, x_n$ are *input variables*, and $\Box \in \{\wedge, \vee, \Leftrightarrow\}$, from the CNF formula using a *graph of clauses* where i) each node corresponds to a clause, ii) each edge corresponds to a pair of clauses exhibiting a resolvent clause, and iii) each edge is labeled either by $\mathbb{T}$ when the resolvent is tautological or $\mathbb{R}$ when the resolvent is not tautological. For instance, an equation $a = b \Leftrightarrow c$ (equivalent to $a = b \oplus c \oplus \top$) corresponds to a clique graph of four clauses where all edges are labeled with $\mathbb{T}$. Such a graph is not explicitly represented but equations are extracted by checking for each clause which clauses exhibit tautological resolvents with it. Any variable in an equation of the form $y = x_1 \Leftrightarrow \cdots \Leftrightarrow x_n$ can take the role of the output variable, so for such equations, the output variable is selected among the set of output variables already defined for other equations. After the equations have been extracted, the CNF formula can be simplified with them. Parity reasoning is performed when the $\Leftrightarrow$ equations (xor-constraints) are used to eliminate variables using two properties: i) $(a \Leftrightarrow a \Leftrightarrow b_1 \Leftrightarrow \cdots \Leftrightarrow b_n)$ is equivalent to $(b_1 \Leftrightarrow \cdots \Leftrightarrow b_n)$, and ii) $(l \Leftrightarrow A_1), (l \Leftrightarrow A_2), \ldots, (l \Leftrightarrow A_m)$ is satisfiable iff $(A_1 \Leftrightarrow A_2), \ldots, (A_{m-1} \Leftrightarrow A_m)$ is satisfiable. In our work, we use an *incidence graph*, a graph where each xor-constraint and variable has a corresponding node and there is an edge between an xor-constraint node and a variable node if the variable occurs in the xor-constraint, i) in Chapter 5, to perform approximating tests to detect whether unit propagation or equivalence reasoning is enough to deduce all xor-implied literals, ii) in Chapter 6, to split xor-constraint conjunctions into components that can be handled individually, and iii) in Chapter 7, to compute EC- and

GE-simulation formulas to simulate stronger parity reasoning with unit propagation.

2.6 Benchmarks

Here we introduce the benchmarks we use to evaluate the methods presented in this thesis. The benchmark instances consist of CNF-xor encodings of known-plaintext attacks on different encryption ciphers and SAT Competition instances that contain (CNF-translations of) xor-constraints. The task in the instances based on cryptographic ciphers is to recover a suitable key that generates the given prefix of the keystream. The initial value (IV) vector is randomly generated and given in the problem instances. As there are far fewer generated keystream bits than key bits, a number of keys probably produce the same prefix of the keystream. Thus, all cipher-based instances are satisfiable.

A5/1. The stream cipher A5/1, analyzed in Biham and Dunkelman [2000], was used to encrypt communication in the GSM cellular telephone standard. It has 64 bits of internal state in three linear feedback shift registers. The task is to recover a suitable 64-bit key that generates the given 1 to 20 first keystream bits. These instances have 27000–33000 clauses and 1200–1500 xor-constraints in up to 321 xor-clusters. The xor-clusters have very few (1 to 9) xor-constraints except for one bigger xor-cluster (over 600 xor-constraints).

DES. The block cipher DES [1977] has a block size of 64 bits and uses a key whose effective length is 56 bits. We modeled the known-plaintext attack on a reduced configuration of DES with 4 rounds on 2 blocks. The instances have 27000–28400 clauses and 31–320 xor-constraints in up to 24 xor-clusters. On these instances only around 1% of the constraints are xor-constraints and the xor-constraints are furthermore partitioned into small xor-clusters separated by large number of clauses.

FEAL. The block cipher FEAL-N described in Miyaguchi [1991] has a block size of 64 bits and uses a 64-bit key. We modeled the known-plaintext attack and the chosen-plaintext attack on a few reduced configurations of FEAL ranging from 2 to 4 rounds on 2-10 blocks. The CNF-part has 2800-13700 variables in 2200-11400 clauses. The xor-part consists of two xor-clusters where each has 1400-7200 variables in 600-3400 xor-constraints.

Grain. The stream cipher Grain presented in Hell *et al.* [2007] has 160 bits of internal state consisting of one 80-bit linear feedback shift register and one 80-bit non-linear feedback shift register. It uses an 80-bit key and a 64-bit IV with 160 initialization rounds. The task in is to recover a suitable key that generates the given prefix of 1 to 20 first keystream bits. These instances have 9800–10400 clauses and a single xor-cluster consisting of 6400–6800 xor-constraints

Hitag2. The stream cipher Hitag2, analyzed in Courtois *et al.* [2009], is widely used in RFID car locks in the automobile industry. It has 48 bits of internal state in a 48-bit linear feedback shift register and a non-linear function with 20 inputs. The task is to recover a suitable 48-bit key that generates the given prefix of 33 to 38 first keystream bits. These instances have 6700–7300 clauses and 3700–4100 xor-constraints in a single xor-cluster.

SAT Competition 2005 - 2011 Instances. We applied the xor-constraint extraction algorithm described in Soos [2010] to the benchmark instances in “crafted” and “industrial/application” categories of the SAT Competitions 2005, 2007, 2009, and 2011 (available at `http://www.satcompetition.org/`) and found a large number of instances with xor-constraints. The “trivial” unary and binary xor-constraints were eliminated by unit propagation and substitution, respectively, leaving 474 instances with xor-constraints, with some duplicates due to overlap in the competitions. The instances have 0–3130000 clauses and 1–330000 xor-constraints in up to 6900 xor-clusters. Figure 2.2 shows the number of xor-constraints with respect to the number of clauses, and Figure 2.3 the number of variables in xor-constraints with respect to the number of variables in CNF-part. There are 103 instances consisting solely of xor-constraints with 10–326000 xor-constraints in one xor-cluster. Figure 2.4 shows the number of xor-clusters.

Trivium. The stream cipher Trivium presented in Cannière [2006] has 288 bits of internal state consisting of three shift registers of different lengths. The known-plaintext attack is modeled by generating a small number (ranging from one to twenty in our experiments) of keystream bits after 1152 initialization rounds. The instances have 8000–8600 clauses and 7100–8800 xor-constraints in 2–3 xor-clusters.

To illustrate how an application-level problem can be translated to a CNF-xor formula, we describe in more detail the structure of Trivium and the encoding of the known-plaintext attack. Figure 2.6 shows the structure of Trivium. In each round, each of the three shift registers $s_{1,\ldots,93}$, $s_{94,\ldots,177}$, and $s_{178,\ldots,288}$ is shifted with one bit and updated with a non-linear combination of select bits. One output bit z_i is produced per round.

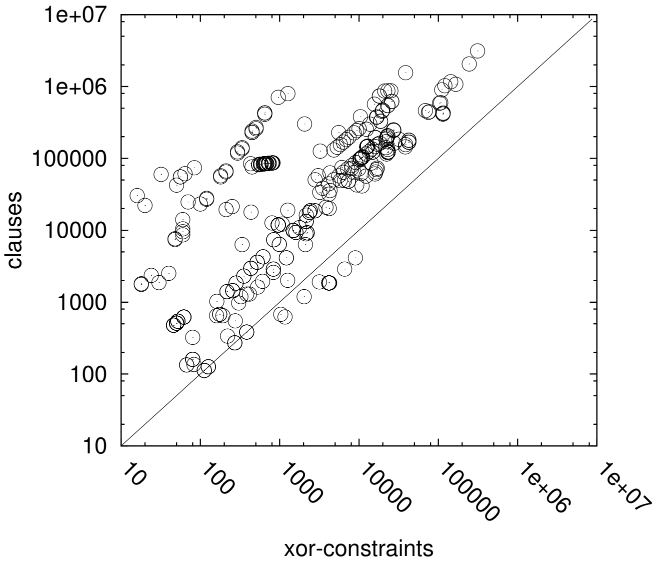


Figure 2.2. Number of xor-constraints vs clauses in SAT Competition instances

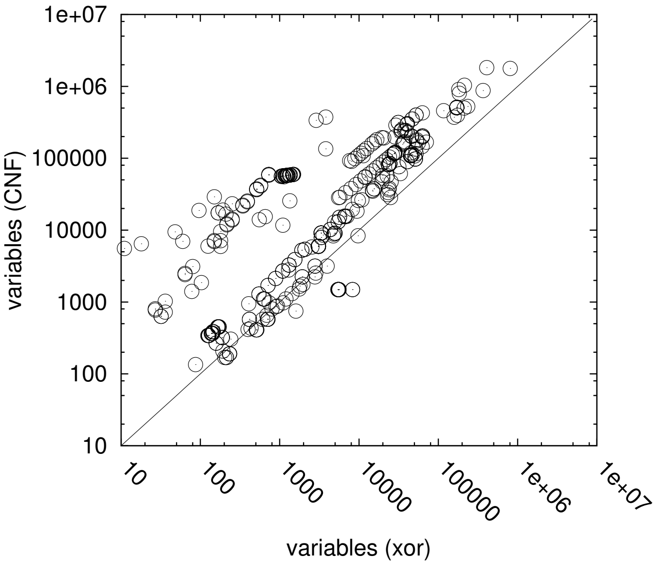


Figure 2.3. Number of variables in xor-constraints vs variables in CNF in SAT Competition instances

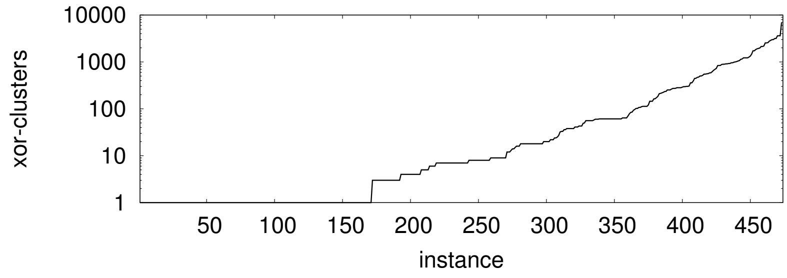


Figure 2.4. Number of xor-clusters in SAT Competition instances

```
1.  for $i = 1$ to $N$:
2.      $t_1 \leftarrow s_{66} + s_{93}$
3.      $t_2 \leftarrow s_{162} + s_{177}$
4.      $t_3 \leftarrow s_{243} + s_{288}$
5.      $z_i \leftarrow t_1 + t_2 + t_3$
6.      $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
7.      $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
8.      $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
9.      $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
10.     $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
11.     $(s_{178}, s_{279}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
```

Figure 2.5. Pseudo-code for Trivium where "+" and "·" operations stand for addition and multiplication over GF(2), i.e., XOR and AND

A complete description of the cipher is given by the pseudo-code in Figure 2.5. The 288-bit initial state is initialized with an 80-bit key $K_{1,\ldots,80}$ and an 80-bit IV $IV_{1,\ldots,80}$ as follows:

$$\begin{aligned}
(s_1, s_2, \ldots, s_{93}) &\leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0) \\
(s_{94}, s_{95}, \ldots, s_{177}) &\leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots, 0) \\
(s_{178}, s_{179}, \ldots, s_{288}) &\leftarrow (0, \ldots, 0, 1, 1, 1)
\end{aligned}$$

The internal state is then updated 1152 rounds as described in Figure 2.5 but without producing any output bits. The Trivium benchmark instances, that is, CNF-xor formulas modeling the known-plaintext attack are created by first generating a textual description of a Boolean circuit modeling the cipher. The pseudo-code in Figure 2.5 can be used with few modifications to output such a Boolean circuit. Figure 2.7 shows a part of the Boolean circuit which contains the gate for the first output bit z_1.

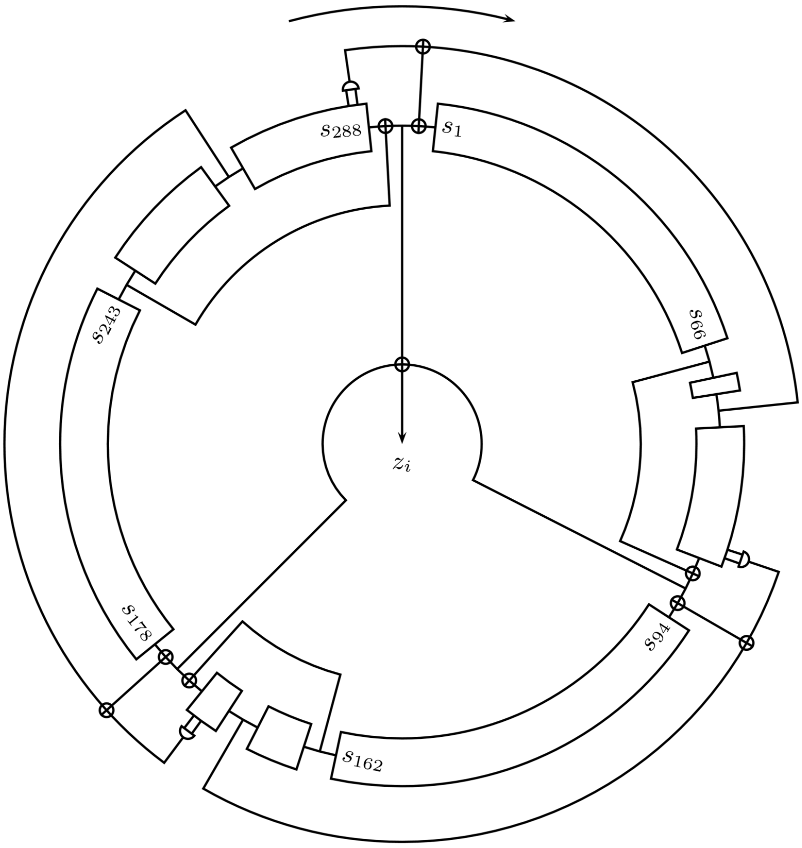


Figure 2.6. Structure of Trivium (from Cannière [2006])

```
...
z1 := ODD(_t1_1153,_t2_1153,_t3_1153);
_s1_1153 := ODD(_t3_1153,AND(_s178_1044,_s178_1043),_s1_1084);
_s94_1153 := ODD(_t1_1153,AND(_s1_1062,_s1_1061),_s94_1075);
_s178_1153 := ODD(_t2_1153,AND(_s94_1071,_s94_1070),_s178_1066);
_t1_1154 := ODD(_s1_1088,_s1_1061);
_t2_1154 := ODD(_s94_1085,_s94_1070);
_t3_1154 := ODD(_s178_1088,_s178_1043);
...
```

Figure 2.7. Partial Boolean circuit for Trivium in format accepted by BCpackage

The Boolean circuit is then translated to a CNF-xor formula using a tool called BCpackage[3] by Tommi Junttila. The tool performs some simplifications such as detecting common subexpressions and propagating constraint values.

[3]https://users.ics.aalto.fi/tjunttil/circuits/index.html

3. XOR-Deduction Systems

An xor-reasoning module employs an *xor-deduction system* (i) to check whether the original xor-conjunction ϕ_{xor} is still satisfiable with respect to the xor-assumptions $l_1 \wedge \cdots \wedge l_k$, and (ii) to deduce *xor-implied* literals implied by $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$. As explained in Section 2.4, checks to detect an *xor-conflict*, the unsatisfiability of $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$, can be incomplete provided that a definite answer can be given when all the variables have been assigned with xor-assumptions. Xor-implied literal detection can also be incomplete, and often is to avoid computational overhead. In this chapter, we present and experimentally evaluate four different xor-deduction systems that can be used in an xor-reasoning module:

- UP: a simple xor-deduction system capable of plain unit propagation over xor-constraints presented in [II]
- Subst: an xor-deduction system capable of equivalence reasoning with substitution-based inference rules presented in Laitinen *et al.* [2010]
- EC: an xor-deduction system capable of equivalence reasoning with equivalence class-based reasoning presented in [I]
- IGJ: an xor-deduction system that provides complete parity reasoning using incremental Gauss-Jordan elimination presented in [IV]

3.1 Unit Propagation (UP)

The first xor-deduction system UP only performs plain unit propagation on xor-constraints. An advantage of using it over CNF-based unit propagation is that unit propagation can be performed on an xor-constraint D involving many variables without a CNF-translation which i) is expo-

$$\oplus\text{-Unit}^{+}\colon \frac{x \equiv \top \quad C}{C\,[x/\top]} \qquad \oplus\text{-Unit}^{-}\colon \frac{x \equiv \bot \quad C}{C\,[x/\bot]}$$

Figure 3.1. Inference rules of UP; the symbol x is variable and C is an xor-constraint involving x

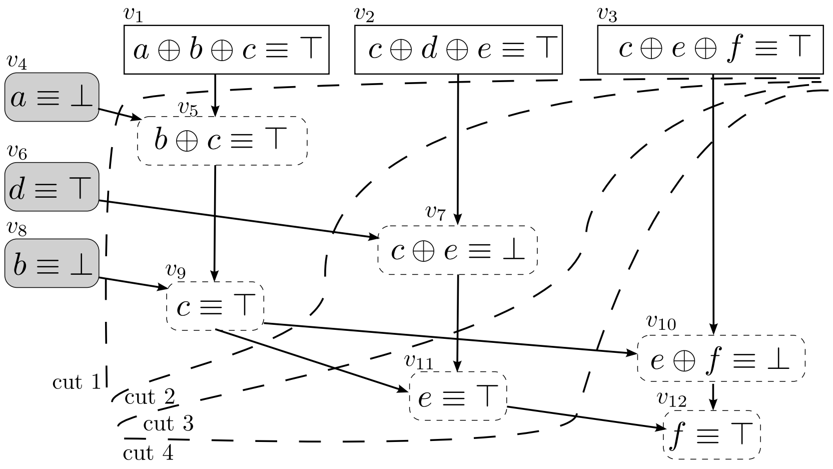


Figure 3.2. A UP-derivation

nential (the straightforward CNF-translation $\text{cnf}(D)$), or ii) involves new auxiliary variables and additional clauses. It also serves as a basis for our parity-based xor-implied literal explanation techniques developed later in Chapter 4.

To deduce xor-implied literals and xor-conflicts on the conjunction $\psi = \phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$, the UP xor-deduction systems applies the inference rules in Figure 3.1. Formally, the state of the xor-deduction system UP is represented by a UP-*derivation* which is a finite, vertex-labeled directed acyclic graph $G = \langle V, E, L\rangle$, where each vertex $v \in V$ is labeled with an xor-constraint $L(v)$ and the following holds for each vertex v:

1. v has no incoming edges (i.e. v is an *input vertex*) and $L(v)$ is an xor-constraint in ψ, or

2. v has two incoming edges originating from vertices v_1 and v_2, and $L(v)$ is derived from $L(v_1)$ and $L(v_2)$ using one of the inference rules.

As an example, Figure 3.2 shows a UP-derivation for $\phi_{\text{xor}} \wedge (a \equiv \bot) \wedge (d \equiv \top) \wedge (b \equiv \bot)$, where $\phi_{\text{xor}} = (a \oplus b \oplus c \equiv \top) \wedge (c \oplus d \oplus e \equiv \top) \wedge (c \oplus e \oplus f \equiv \top)$ (please ignore the "cut" lines for now). An xor-constraint C is UP-*derivable* on ψ, denoted by $\psi \vdash_{\text{UP}} C$, if there exists a UP-derivation on ψ that contains a vertex labeled with C. In Figure 3.2, the literal f is UP-derivable and the xor-deduction system UP deduces f as an xor-implied literal after $\neg a$, d, and $\neg b$ are given as xor-assumptions. A direct consequence of the definition of UP-derivations and the soundness of the inference rules is that if an UP-derivation on ψ contains a vertex labeled with the xor-constraint C, then C is a logical consequence of ψ, i.e. $\psi \vdash_{\mathsf{UP}} C$ implies $\psi \models C$. A UP-derivation on ψ is a UP-*refutation of* ψ if it contains a vertex labeled with the false literal $\bot$; in this case, ψ is unsatisfiable. A UP-derivation G on ψ is *saturated* if for each unary xor-constraint C such that $\psi \vdash_{\mathsf{UP}} C$ there is a vertex v in G with the label $L(v) = C$. Note that UP is not refutationally complete, e.g. there is no UP-refutation of the unsatisfiable conjunction $(a \oplus b \equiv \top) \wedge (a \oplus b \equiv \bot)$. However, it is "eventually refutationally complete" in the DPLL(XOR) setting: if each variable in ψ occurs in a unary xor-constraint in ψ, then the empty xor-constraint is UP-derivable iff ψ is unsatisfiable; thus when the CDCL SAT solver has assigned a value to all variables in ϕ_{xor}, the UP-module can check whether all the xor-constraints are satisfied.

To satisfy the requirements for the EXPLAIN method in the DPLL(XOR) framework, the xor-reasoning module and the underlying xor-deduction system must be able to provide an *implying clause* for each xor-implied literal.

The implying clause for an xor-implied literal can be computed by interpreting the rules $\oplus$-Unit$^+$ and $\oplus$-Unit$^-$ as implications:

$$(x \equiv \top) \wedge C \quad \Rightarrow \quad C\,[x/\top] \tag{3.1}$$

$$(x \equiv \bot) \wedge C \quad \Rightarrow \quad C\,[x/\bot] \tag{3.2}$$

respectively, and recursively rewriting the xor-implied literal with the left-hand side conjunction until a certain cut of the UP-derivation is reached. Formally, a *cut* of a UP-derivation $G = \langle V, E, L\rangle$ is a partitioning $(V_{\text{a}}, V_{\text{b}})$ of V. A *cut for a non-input vertex* $v \in V$ is a cut $(V_{\text{a}}, V_{\text{b}})$ such that (i) $v \in V_{\text{b}}$, and (ii) if $v' \in V$ is an input vertex and there is a path from v' to v, then $v' \in V_{\text{a}}$. Now assume a UP-derivation $G = \langle V, E, L\rangle$ for $\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_k$. For each non-input node v in G, and each cut $W = \langle V_{\text{a}}, V_{\text{b}}\rangle$ of G for v, the *implicative explanation* of v under W is the conjunction $\mathit{Expl}(v, W) = f_W(v)$, there f_W is recursively defined as follows:

E1 If u is an input node with $L(u) \in \phi_{\text{xor}}$, then $f_W(u) = \bot \equiv \bot$.

E2 If u is an input node with $L(u) \in \{l_1, ..., l_k\}$, then $f_W(u) = L(u)$.

E3 If u is a non-input node in V_{a}, then $f_W(u) = L(u)$.

E4 If u is a non-input node in V_{b}, then $f_W(u) = f_W(u_1) \wedge f_W(u_2)$, where u_1 and u_2 are the source nodes of the two edges incoming to u.

Based on Equations (3.1) and (3.2) we can easily conclude that $\phi_{\text{xor}} \models \mathit{Expl}(v, W) \Rightarrow L(v)$.

The implicative explanation $\mathit{Expl}(v, W)$ was originally defined in Laitinen *et al.* [2010] for the cut W as follows: $\mathit{Expl}(v, W) = \bigwedge_{u \in \text{reasons}(W)} L(u)$, where $\text{reasons}(W) = \{u \in V_{\text{a}} \mid L(u) \notin \phi_{\text{xor}} \wedge \exists u' \in V_{\text{b}} : \langle u, u' \rangle \in E\}$ is the *reason set* for W. A cut W is *CNF-compatible* if $L(u)$ is a unary xor-constraint for each $u \in \text{reasons}(W)$. Thus if the cut W is CNF-compatible, then $\mathit{Expl}(v, W) \Rightarrow L(v)$ is the required clause implying the xor-implied literal $L(v)$ provided that $L(v)$ is a literal and not a longer xor-constraint.

Example 1. *Consider again the* UP*-derivation on* $\phi_{\text{xor}} \wedge (a \equiv \bot) \wedge (d \equiv \top) \wedge (b \equiv \bot)$ *in Figure 3.2. It has four cuts, 1–4, for the vertex* v_{12}*, corresponding to the implicative explanations* $\neg a \wedge d \wedge \neg b$*,* $c \wedge d$*,* $c \wedge (c \oplus e \equiv \bot)$*, and* $e \wedge c$*, respectively. The non-CNF-compatible cut 3 cannot be used to give an implying clause for the xor-implied literal* f *but the others can; the one corresponding to the cut 2 is* $(\neg c \vee \neg d \vee f)$*.* ♣

The UP-derivation bears an important similarity with "traditional" implication graph of a SAT solver where each vertex represents a variable assignment: graph partitions are used to derive clausal explanations for implied literals. Different partitioning schemes for such implication graphs have been studied in Zhang *et al.* [2001], and we can directly adopt some of them for our analysis. A cut $W = (V_{\text{a}}, V_{\text{b}})$ for a non-input vertex v is:

1. *closest cut* if W is the CNF-compatible cut with the smallest possible V_{b} part. Observe that each implying clause derived from these cuts is implied by a single original xor-constraint; e.g., $(\neg c \vee \neg e \vee f)$ obtained from the cut 4 in Figure 3.2 is implied by $(c \oplus e \oplus f \equiv \top)$ as $(\neg c \vee \neg e \vee f)$ is a clause in $\text{cnf}(c \oplus e \oplus f \equiv \top)$ and $(c \oplus e \oplus f \equiv \top)$ is an original xor-constraint.

2. *furthest cut* if V_{b} is maximal. Note that furthest cuts are also CNF-compatible as their reason sets consist only of xor-assumptions.

In the implementation of the xor-deduction system UP, we use a modified version of the 2-watched literals scheme first presented in Moskewicz

$$\oplus\text{-Unit}^{+}\colon \frac{x \equiv \top \quad C}{C\,[x/\top]} \qquad \oplus\text{-Eqv}^{+}\colon \frac{x_1 \oplus x_2 \equiv \bot \quad C}{C\,[x_1/x_2]}$$

$$\oplus\text{-Unit}^{-}\colon \frac{x \equiv \bot \quad C}{C\,[x/\bot]} \qquad \oplus\text{-Eqv}^{-}\colon \frac{x_1 \oplus x_2 \equiv \top \quad C}{C\,[x_1/(x_2 \oplus \top)]}$$

Figure 3.3. Inference rules of Subst; the symbols x, x_1, and x_2 are variables while C is a xor-constraint

et al. [2001] for clauses; all but one of the *variables* in an xor-constraint need to be assigned before the xor-constraint implies the last one. Thus it suffices to have two *watched variables*. MoRsat by Chen [2009] uses a similar data structure for all clauses and has 2×2 watched literals for an xor-constraint. Cryptominisat by Soos *et al.* [2009] uses a scheme similar to ours except that it manipulates the polarities of literals in an xor-constraint while we take the polarities into account in the explanation phase. Because of this implementation technique, the implementation does not explicitly represent the non-unary non-input vertices in UP-derivations.

3.2 Equivalence Reasoning by Substitution (SUBST)

The xor-deduction system Subst presented in Laitinen *et al.* [2010] is conceptually very similar to the xor-deduction system UP presented in Section 3.1. It naturally supports plain unit propagation over xor-constraints, but it is also capable of equivalence reasoning by allowing substitutions of variables occurring in binary xor-constraints.

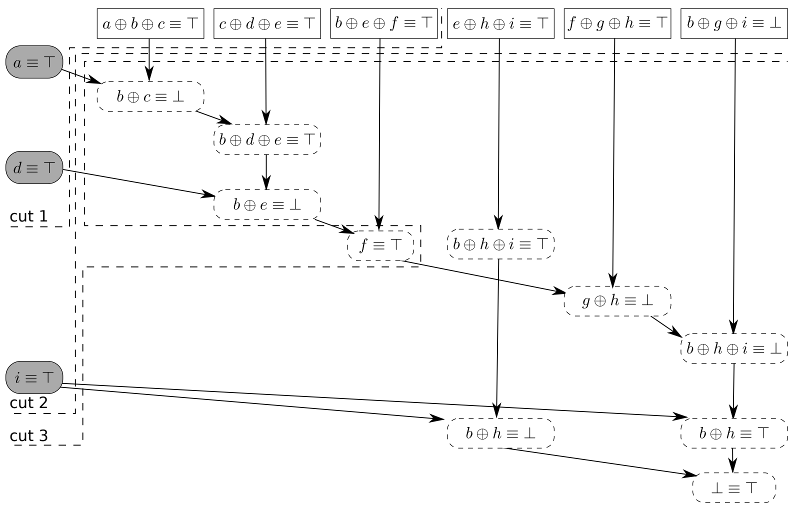
The xor-deduction system Subst consists of the inference rules in Figure 3.3. Except for the two additionally allowed inference rules $\oplus$-Eqv^{+} and $\oplus$-Eqv^{-}, Subst-*derivation* is defined similarly to UP-derivation.

Example 2. *Assume a conjunction of xor-constraints*

$$\begin{aligned}\phi_{\text{xor}} \;=\;& (a \oplus b \oplus c \equiv \top) \wedge (c \oplus d \oplus e \equiv \top) \wedge (b \oplus e \oplus f \equiv \top) \wedge \\ & (e \oplus h \oplus i \equiv \top) \wedge (f \oplus g \oplus h \equiv \top) \wedge (b \oplus g \oplus i \equiv \bot).\end{aligned}$$

Figure 3.4 shows a Subst*-refutation of* $\phi_{\text{xor}} \wedge (a \equiv \top) \wedge (d \equiv \top) \wedge (i \equiv \top)$. *The literal* f *is* Subst*-derivable from* $\phi_{\text{xor}} \wedge (a \equiv \top) \wedge (d \equiv \top)$*; the xor-deduction system* Subst *deduces* f *as an xor-implied literal on* ϕ_{xor} *after* a *and* d *are given as xor-assumptions. Observe that* f *is* not UP*-derivable from* $\phi_{\text{xor}} \wedge (a \equiv \top) \wedge (d \equiv \top)$*, i.e.* Subst *is a stronger deduction system than*

UP *in this sense.* ♣.



$a \oplus b \oplus c \equiv \top$
$c \oplus d \oplus e \equiv \top$
$b \oplus e \oplus f \equiv \top$
$e \oplus h \oplus i \equiv \top$
$f \oplus g \oplus h \equiv \top$
$b \oplus g \oplus i \equiv \bot$
$a \equiv \top$
$b \oplus c \equiv \bot$
$b \oplus d \oplus e \equiv \top$
$d \equiv \top$
$b \oplus e \equiv \bot$
cut 1
$f \equiv \top$
$b \oplus h \oplus i \equiv \top$
$g \oplus h \equiv \bot$
$b \oplus h \oplus i \equiv \bot$
$i \equiv \top$
cut 2
cut 3
$b \oplus h \equiv \bot$
$b \oplus h \equiv \top$
$\bot \equiv \top$

Figure 3.4. A Subst-derivation

Related work. The xor-deduction system Subst has similarities to other systems. The solver 2cls+eq Bacchus [2002] performs all possible resolutions of pairs of binary clauses in addition to basic unit propagation. This system in 2cls+eq is denoted by BinRes. For example, given a conjunction of clauses $\phi_{\text{or}} = (a \vee b) \wedge (\neg a \wedge c) \wedge (\neg b \vee c)$, it can deduce that the variable c must be true. The xor-deduction system Subst cannot deduce this because the formula ϕ_{or} cannot be translated to a logically equivalent conjunction of xor-constraints. However, Subst can deduce xor-constraints whose CNF-translations cannot be derived by BinRes. For example, consider an xor-constraint conjunction $\phi_{\text{xor}} = (a \oplus b \equiv \top) \wedge (a \oplus b \oplus c \equiv \bot)$. The xor-constraint $(c \equiv \top)$ is Subst-derivable on ϕ_{xor}, i.e., $\phi_{\text{xor}} \vdash_{\text{Subst}} (c \equiv \top)$, but BinRes cannot deduce the unary clause (c) from $\text{cnf}(\phi_{\text{xor}})$. The solver 2cls+eq in fact uses a stronger system, HypBinRes, which performs a hyper-resolution step (a resolution step on more than two clauses) on one n-ary clause $(l_1 \vee l_2 \vee \ldots l_n)$, where $n \geq 2$, and $n-1$ binary clauses each of the form $(\neg l_i \vee l)$, where $i \in \{1, \ldots, n-1\}$. Again, as HypBinRes subsumes BinRes, Subst cannot reproduce deductions on clauses that do not have representations as xor-constraints, but Subst can still deduce xor-constraints whose CNF-translations HypBinRes cannot derive, e.g. HypBinRes either cannot deduce the unary clause (c) from $\text{cnf}(\phi_{\text{xor}})$. The third system, EqReduce, in 2cls+eq performs an "equality reduction" meaning that if a formula ϕ_{or} contains two clauses $(\neg a \vee b)$ and $(a \vee \neg b)$, that is $\text{cnf}(a \oplus b \equiv \top)$, for some variables $a, b \in \text{vars}(\phi_{\text{or}})$, then i) all instances of the variable b are replaced in ϕ_{or} by a (or vice versa), ii) all clauses that now contain both a and $\neg a$ are removed, iii) and duplicate instances of a and $\neg a$ are removed from all clauses. The system EqReduce subsumes the inference rules $\oplus$-Eqv$^+$ and $\oplus$-Eqv$^-$ in the sense that if $\phi_{\text{xor}} \vdash_{\text{Subst}} D$ for some formulas ϕ_{xor} and D, then combined with HypBinRes and basic unit propagation it can deduce the clauses in $\text{cnf}(D)$ from the formula $\text{cnf}(\phi_{\text{xor}})$.

The solver march_eq in Heule and van Maaren [2004] uses binary xor-constraints in a preprocessing step to eliminate variable occurrences in other xor-constraints in a way similar to EqReduce and the inference rules $\oplus$-Eqv$^+$ and $\oplus$-Eqv$^-$.

The solver EqSatz by Li [2000b,a] has six inference rules to deal with "equivalency clauses", that is, xor-constraints. The inference rules of EqSatz are shown in Figure 3.5. The solver EqSatz uses deduced binary xor-constraints to eliminate variable occurrences in xor-constraints in addition to the inference rules in Figure 3.5.

l_1	$l_1 \leftrightarrow l_2 \leftrightarrow l_3$	$\leadsto$	$l_2 \leftrightarrow l_3$
$\neg l_1$	$l_1 \leftrightarrow l_2 \leftrightarrow l_3$	$\leadsto$	$\neg(l_2 \leftrightarrow l_3)$
$l_1 \leftrightarrow l_1 \leftrightarrow l_2$		$\leadsto$	l_2
$l_1 \leftrightarrow l_2 \leftrightarrow l_3$	$l_1 \leftrightarrow l_2 \leftrightarrow l_4$	$\leadsto$	$l_3 \leftrightarrow l_4$
$l_1 \rightarrow (l_3 \leftrightarrow l_4)$	$\neg l_1 \rightarrow (l_3 \leftrightarrow l_4)$	$\leadsto$	$l_3 \leftrightarrow l_4$
$l_1 \rightarrow (l_3 \leftrightarrow l_4)$	$\neg l_1 \rightarrow (\neg l_3 \leftrightarrow l_4)$	$\leadsto$	$l_1 \leftrightarrow l_3 \leftrightarrow l_4$

Figure 3.5. Inference rules of EqSatz. When the clause(s) on the left side are deduced, the right side can deduced.

The last two inference rules in Figure 3.5 are also used in a look-ahead branching step to deduce and learn new binary and ternary xor-constraints. When solving the formula ϕ_{or}, each non-assigned variable $x \in \text{vars}(\phi_{\text{or}})$ is examined by performing experimental unit propagations to see the impact of branching on x with both truth values giving two simplified formulas ϕ'_{or} and ϕ''_{or}. All new binary and ternary xor-constraints that belong both to ϕ'_{or} and to ϕ''_{or} are added in the formula ϕ_{or}.

Provided that each xor-constraint in ϕ_{xor} has at most three variables, if Subst can deduce an xor-constraint D from ϕ_{xor}, then it is possible to deduce the clauses in $\text{cnf}(D)$ from $\text{cnf}(\phi_{\text{xor}})$ using the inference rules of EqSatz. The converse is not true; for instance, Subst cannot combine two ternary xor-constraints to deduce a new binary xor-constraint. However, such reasoning could be performed to some extent in a preprocessing step.

$$\frac{x \equiv \top \quad x \equiv \bot}{\bot}$$

(a) Conflict

$$\frac{x \equiv p_1 \quad x \oplus y \oplus z \equiv p_2}{y \oplus z \equiv p_1 \oplus p_2}$$

(b) $\oplus$-Unit^3

$$\frac{x \equiv p_1 \quad x \oplus y \equiv p_2}{y \equiv p_1 \oplus p_2}$$

(c) $\oplus$-Unit^2

$$\frac{x_1 \oplus x_2 \equiv p_1 \quad \ldots \quad x_{n-1} \oplus x_n \equiv p_{n-1} \quad x_1 \oplus x_n \oplus y \equiv p}{y \equiv p_1 \oplus p_2 \oplus \ldots \oplus p_{n-1} \oplus p}$$

(d) $\oplus$-Imply

$$\frac{\begin{array}{c} x_1 \oplus x_2 \equiv p_1 \quad \ldots \quad x_{n-1} \oplus x_n \equiv p_{n-1} \quad x_n \oplus x_1 \equiv p_n \\ \text{provided that } p_1 \oplus \ldots \oplus p_n = \top \end{array}}{\bot}$$

(e) $\oplus$-Conflict

Figure 3.6. Inference rules of EC; the symbols x, x_i, y, z are all variables while the p_i symbols are constants $\bot$ or $\top$.

Also, since EqSatz considers only xor-constraints with at most three variables, Subst can perform deductions on longer xor-constraints that EqSatz cannot deduce. For instance, $(a \oplus b \equiv \bot) \wedge (a \oplus b \oplus c \oplus d \equiv \top) \vdash_{\textsf{Subst}} (c \oplus d \equiv \top)$, but EqSatz cannot deduce the clauses in $\mathrm{cnf}(c \oplus d \equiv \top)$ from the clauses in $\mathrm{cnf}(a \oplus b \equiv \bot) \wedge \mathrm{cnf}(a \oplus b \oplus c \oplus d \equiv \top)$.

3.3 Equivalence Reasoning using Equivalence Classes (EC)

The EC xor-deduction system presented in [I] supports equivalence reasoning by manipulating equivalence classes of literals rather than rewriting xor-constraints through substitutions. The deduction power of EC is shown to be equivalent to Subst when considering deducible xor-implied literals and xor-conflicts, but can give shorter implicative explanations for xor-implied literals. The inference rules of the Subst xor-deduction system are relatively efficiently implementable but it is hard to detect when the substitution rules ($\oplus$-Eqv$^+$ and $\oplus$-Eqv$^-$) should be applied. A straightforward way to make sure all deducible xor-implied literals and xor-conflicts are detected is to eliminate all variables occurrences of the other variable in a binary xor-constraint by using the substitution rules. However, it is not guaranteed that all of the derived xor-constraints are used when deducing other xor-implied literals or xor-conflicts, and even if they are, some of the substitutions may still be unnecessary. The aim in EC xor-deduction system is to avoid deriving unnecessary xor-constraints.

For implementation efficiency reasons, we split xor-constraints with four or more variables to an equisatisfiable conjunction of xor-constraints with at most three variables by using auxiliary variables. This is done by repeatedly applying the rewrite rule

$$(x_1 \oplus x_2 \oplus \ldots \oplus x_n \equiv p) \rightsquigarrow (x_1 \oplus x_2 \oplus y \equiv \bot) \wedge (y \oplus x_3 \oplus x_4 \oplus \ldots \oplus x_n \equiv p)$$

where y is a new variable not occurring in other clauses and $p \in \mathbb{B}$. However, the xor-deduction system Subst may not be able to deduce all xor-implied literals after the translation; e.g. $(a \oplus c \equiv \bot) \wedge (a \oplus b \oplus c \oplus d \equiv \bot) \wedge (b \oplus d \oplus e \equiv \top) \vdash_{\mathsf{Subst}} e \equiv \top$, but $(a \oplus c \equiv \bot) \wedge (a \oplus b \oplus y \equiv \bot) \wedge (y \oplus c \oplus d \equiv \bot) \wedge (b \oplus d \oplus e \equiv \top) \nvdash_{\mathsf{Subst}} e \equiv \top$. The inferences rules of the EC xor-deduction system are listed in Figure 3.6. We define EC-*derivation*, EC-*derivable* (denoted by $\psi \vdash_{\mathrm{EC}} C$), EC-*refutation*, and (CNF-compatible) cut for a non-input vertex in EC-derivation as in Section 3.1 except that in an EC-derivation a vertex may have more than two incoming edges.

The $\oplus$-Unit^3 and $\oplus$-Unit^2 rules perform unit propagation, producing an equivalence ($\oplus$-Unit^3) or a unary clause ($\oplus$-Unit^2).

The $\oplus$-Imply and $\oplus$-Conflict rules propagate known equivalences, producing either (i) an unary clause if the equivalences force literals in an original xor-constraint to have the same/opposite value, or (ii) a conflict if the equivalences force a variable to have a value opposite to its value, i.e. if $x \equiv \neg x$ should hold.

Example 3. *Consider again the formula ϕ_{xor} in Example 2. Figure 3.7 shows an* EC*-refutation of $\phi_{\mathrm{xor}} \wedge (a \equiv \top) \wedge (d \equiv \top) \wedge (i \equiv \top)$. Comparing to the* Subst*-derivation in Figure 3.4, observe (i) the fewer number of vertices, (ii) n-ary in-degree of vertices caused by the lack of substitution steps and use of $\oplus$-Imply and $\oplus$-Conflict rules, and (iii) that no new ternary xor-constraints are derived. Again, the CNF-compatible cut 2 shows that $\phi_{\mathrm{xor}} \wedge (a \equiv \top) \wedge (d \equiv \top) \wedge (i \equiv \top) \vdash_{\mathrm{EC}} \bot$; thus $\phi_{\mathrm{xor}} \wedge (a \equiv \top) \wedge (d \equiv \top) \wedge (i \equiv \top) \models \bot$ and $\phi_{\mathrm{xor}} \models (\neg a \vee \neg d \vee \neg i)$.* ♣

Despite their differences, the Subst and EC deduction systems are equally powerful with respect to their ability to deduce xor-implied literals and xor-conflicts:

Theorem 2 (Thm. 1 of [I]). *Let ψ be a conjunction of xor-constraints, each of which has at most three variables. For each literal $\hat{l} \in \mathrm{lits}(\psi) \cup \{\bot\}$ the following fact holds: $\psi \vdash_{\mathsf{Subst}} \hat{l}$ iff $\psi \vdash_{\mathrm{EC}} \hat{l}$.*

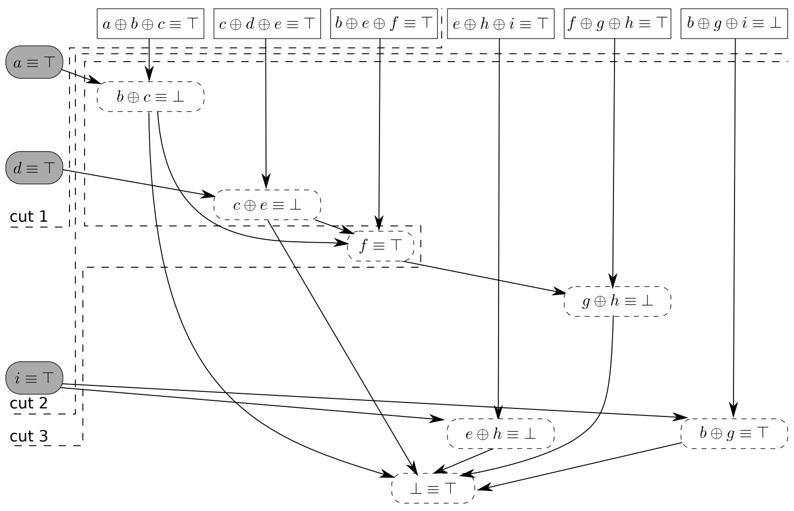


Figure 3.7. An EC-derivation

3.3.1 Implementing with Equivalence Classes

The EC xor-deduction system makes heavy use of binary xor-constraints which can be seen also as equivalences between literals. In the implementation of the EC xor-deduction system, we track and manipulate these equivalence classes of literals in order to efficiently detect when the inference rules are applicable. However, when building an EC-derivation, it is not enough to know which literals are in the same equivalence class; it is also required to tell *why*. To implement this, we augment the equivalence class data structure with an *equivalence reason graph* which is an undirected, edge-labeled graph where (i) the vertex set is $\mathrm{vars}(\phi_{\mathrm{xor}}) \cup \{\top\}$ and (ii) the edge set records the reasons for the equivalence class merge operations. With this data structure there is no need to build an EC-derivation at all since implicative explanations can be directly computed from it.

We illustrate how an equivalence reason graph is used with an example. A more comprehensive description of how equivalence classes are manipulated and how implicative explanations for xor-implied literals are computed in practice can be found in [I].

Example 4. *Consider again the formula ϕ_{xor} in Example 2 and compare the process below to the derivation discussed in Example 3. The full equivalence reason graph constructed is in Figure 3.8 where $\star$ labels xor-assumptions and crossed-over edges denote disequality edges while the normal ones are equality edges.*

When we make the xor-assumption $a \equiv \top$, we first add the edge $\top \overset{\star}{\text{—}} a$

in the equivalence reason graph. The equivalence classes of $\top$ *and* a *are merged giving 8 equivalence classes* $\{\top, a\}, \{d\}, \{f\}, \{b\}, \{c\}, \{e\}, \{g\}$, *and* $\{h\}$. *We then find the xor-constraint* $(a \oplus b \oplus c \equiv \top)$ *including* a *in* ϕ_{xor} *and thus derive the xor-constraint* $b \oplus c \equiv \bot$, *merge the equivalence classes* $\{b\}$ *and* $\{c\}$ *and add the edge* $b \overset{a\,\equiv\,\top}{\text{---}} c$.

Making xor-assumption $d \equiv \top$, *we merge the equivalence classes* $\{\top, a\}$ *and* $\{d\}$, *find the xor-constraint* $(c \oplus d \oplus e \equiv \top)$ *including* d *in* ϕ_{xor} *and thus derive the xor-constraint* $c \oplus e \equiv \bot$, *merge the equivalence classes* $\{b, c\}$ *and* $\{e\}$, *and add the edge* $c \overset{d\,\equiv\,\top}{\text{---}} e$. *After this step, we consider the variables occurring in the* smaller *of the equivalence classes just merged, i.e. the class* $\{e\}$. *We then consider all xor-constraints in* ϕ_{xor} *which include* e *and check whether they also include a variable from the equivalence class* $\{b, c\}$. *Indeed, there is such an xor-constraint* $b \oplus e \oplus f \equiv \top$ *and thus the* $\oplus$-Imply *rule is applicable, allowing us to derive the xor-implied literal* $f \equiv \top$, *so we add the edge* $\top \overset{b\,\oplus\,e\,\equiv\,\bot}{\text{------}} f$. *The classes* $\{f\}$ *and* $\{\top, a, d\}$ *are then merged. After this, there are four equivalence classes* $\{\top, a, d, f\}$, $\{b, c, e\}$, $\{g\}$, *and* $\{h\}$. *Then we derive the xor-constraint* $g \oplus h \equiv \bot$, *merge the equivalence classes* $\{g\}$ *and* $\{h\}$, *and add the edge* $g \overset{f\,\equiv\,\top}{\text{---}} h$. *Finally, after deriving the xor-constraints* $e \oplus h \equiv \bot$ *and* $b \oplus g \equiv \top$ *and adding the edges* $e \overset{i\,\equiv\,\top}{\text{---}} h$ *and* $b \overset{i\,\equiv\,\top}{\not\text{---}} g$, *the equivalence relation becomes inconsistent; thus* $\bot$ *is derived.*

Xor-conflicts manifest themselves in equivalence reason graphs via cyclic paths with an odd number of disequality edges. For instance, in Figure 3.8 there is such a path $b \overset{i\,\equiv\,\top}{\not\text{---}} g \overset{f\,\equiv\,\top}{\text{---}} h \overset{i\,\equiv\,\top}{\text{---}} e \overset{d\,\equiv\,\top}{\text{---}} c \overset{a\,\equiv\,\top}{\text{---}} b$. *Implicative explanations in such cases can be obtained by taking the union*

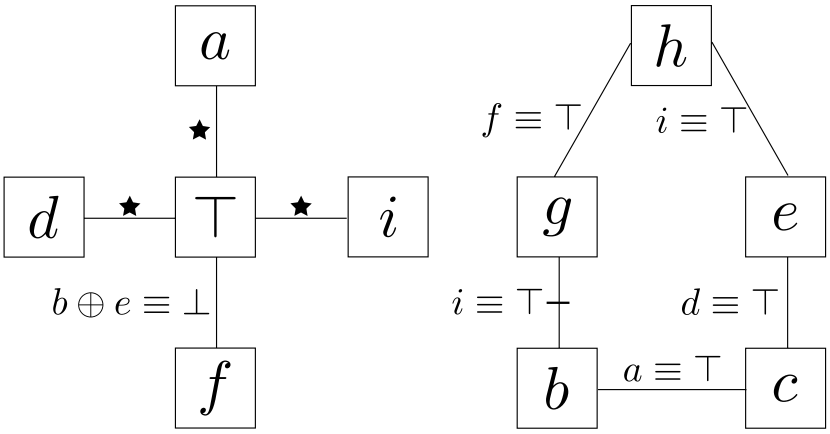


Figure 3.8. An equivalence reason graph

of reasons from the edges in the cycle; in the example, the reason set is $\{a \equiv \top, d \equiv \top, f \equiv \top, i \equiv \top\}$*, corresponding to the "cut 3" in Figure 3.7, and thus the implicative explanation for the xor-conflict is* $(\neg a \vee \neg d \vee \neg f \vee \neg i)$*. Any cyclic path with an odd number of disequality edges can be used but in our implementation we find the shortest such cycle by using breadth-first search. However, since* $f \equiv \top$ *is an xor-implied literal, the SAT solver may need an implying clause for it. The reason from the edge* $\top \overset{b \oplus e \equiv \bot}{\text{———}} f$ *cannot be translated to an implying clause. A suitable explanation can be obtained by taking the union of reasons from any path from* b *to* e *using the version of the equivalence reason graph corresponding to the moment when* $f \equiv \top$ *was derived. As the path* $b \overset{i \equiv \top}{\not\text{—}} g \overset{f \equiv \top}{\text{—}} h \overset{i \equiv \top}{\text{—}} e$ *was added later, the implicative explanation for the xor-implied literal* f *is* $(\neg a \vee \neg d \vee f)$*, corresponding to the "cut 1" in Figure 3.7. Our implementation again uses breadth-first search to find the shortest path. Furthest cuts can be computed by recursively explaining all xor-implied literals until the reason set contains only xor-assumptions.* ♣

Related work. Li [2003] proposes a data structure in the solver EqSatz to represent all equivalent literals in CNF formula. The algorithm to track classes of equivalent literals uses an adaptation of the classical union-find algorithm, as is done in the implementation of EC to track equivalence classes. As no clause learning is used, EqSatz does not track reasons for equivalent literals.

The theorem prover for program checking, Simplify, by Detlefs *et al.* [2005], includes an "E-Graph module" for the theory of equality with uninterpreted function symbols, that is, for literals of the forms $X = Y$ and $X \neq Y$, where X and Y are terms built from variables and applications of uninterpreted function symbols. e.g. $f(f(a, b), b) = c$. An ***E-graph*** is a pair $\langle G, R\rangle$ where i) G is a vertex-labeled directed acyclic multigraph whose nodes represent terms without quantified variables and whose edges are ordered, and ii) R is an equivalence relation on the nodes of G. Figure 3.9 shows an E-graph for the following set of (E-Graph) literals:

$$f(a, b) = a$$
$$f(f(a, b), b) = c$$
$$g(a) \neq g(c)$$

The E-graph used in Simplify is similar to the equivalence reason graph and the associated equivalence relation used in the xor-deduction system EC. The theorem prover Simplify maintains an E-graph to check the sat-

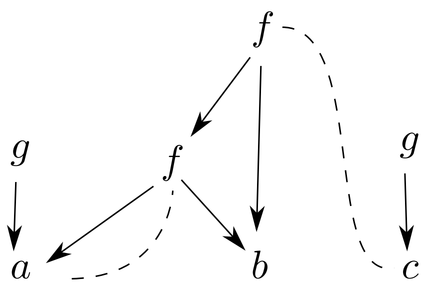


Figure 3.9. An E-graph in used Simplify theorem prover for program checking by Detlefs *et al.* [2005]. The solid arrows represent the structure of terms and the dashed lines represent equivalences between terms. The inequivalence $g(a) \neq g(c)$ is handled separately.

isfiability of a set of (E-graph) literals using incremental algorithms and data structures including a variant of union-find algorithm to maintain the data structure for the equivalence relation. To detect conflicts, the E-graph module also maintains a data structure representing a set of forbidden merge operations. The set is checked before two equivalence classes are merged and in case of a forbidden merge operation, the E-graph module marks the current set of (E-graph) literals as refuted. A notable difference between the E-graph module and the xor-deduction system EC is that the E-graph module does not produce explanations for refuted sets of (E-graph) literals while EC can compute an explanation for an xor-conflict.

Nieuwenhuis and Oliveras [2007] describe an efficient incremental congruence closure algorithm and extend it with a proof-producing union-find data structure. A *congruence* relation is an equivalence relation (reflexive, symmetric, and transitive) which also satisfies the monotonicity axioms $\forall f : (f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)) \Longleftarrow (\forall i \in \{1, \ldots, n\} : a_i = b_i)$. The union-find data structure does not process "redundant" unions (equivalence class merge operations) as the EC xor-deduction system does in the equivalence reason graph, so it produces unique explanations for why two given elements are equivalent in almost optimal time (quasi-linear in the size of the explanation). For comparison, the breadth-first search used in our implementation of EC to find explanations may produce longer explanations if the redundant equivalence class merge operations are not recorded in the equivalence reason graph.

3.4 Incremental Gauss-Jordan Elimination

The xor-deduction system presented in [IV], which we will refer to as IGJ xor-deduction system, is based on incremental Gauss-Jordan elimination and is a complete parity reasoning method in the sense that it can be used to (i) *decide* whether a conjunction of xor-constraints is satisfiable, and (ii) find *all* xor-implied literals and implied equivalences. For comparison, equivalence reasoning systems (such as Subst and EC described in Sections 3.2 and 3.3) cannot always decide the satisfiability or find all xor-implied literals. Similarly, Gaussian elimination, used e.g. in Cryptominisat (version 2.9.2) by Soos *et al.* [2009]; Soos [2010] can decide satisfiability, but does not necessary find all xor-implied literals. The reason for this is that Gaussian elimination translates a system of equations into row echelon form matrix, where all variables except one may have multiple occurrences. For example, consider the row echelon form matrix-like representation:

$$\begin{array}{lllll} a\oplus b & & \oplus d & & \equiv \top \\ & b\oplus c & & \oplus e & \equiv \bot \\ & & c\oplus d & \oplus e & \equiv \top \end{array}$$

for a conjunction ϕ_{xor} of xor-constraints. It is easy to deduce from this that ϕ_{xor} is satisfiable but not that a must always be false, i.e. that $\phi_{\text{xor}} \models a \equiv \bot$. On the other hand, by applying Gauss-Jordan elimination to the same conjunction ϕ_{xor}, we can obtain the reduced row echelon form matrix-like representation:

$$\begin{array}{llll} a & & & \equiv \bot \\ & b & \oplus d & \equiv \top \\ & & c\oplus d\oplus e & \equiv \top \end{array}$$

where we can read directly that $\phi_{\text{xor}} \models a \equiv \bot$.

The xor-deduction system IGJ represents a reduced row echelon form matrix as a set of equations with some requirements:

Definition 1. *A* tableau *for a satisfiable conjunction ϕ_{xor} of xor-constraints is a set $\mathcal{E}$ of equations of form $x_i := x_{i,1} \oplus ... \oplus x_{i,k_i} \oplus p_i$, where x_i is the* basic variable *with exactly one occurrence in $\mathcal{E}$ and $x_{i,1}$,..., x_{i,k_i} are distinct* non-basic *variables in ϕ_{xor} and $p_i \in \mathbb{B}$.*

It is required that $\bigwedge_{x_i:=x_{i,1}\oplus...\oplus x_{i,k_i}\oplus p_i\in\mathcal{E}}(x_i\oplus x_{i,1}\oplus...\oplus x_{i,k_i} \equiv p_i)$ be logically equivalent to ϕ_{xor}.

A conjunction ϕ_{xor} of xor-constraints can be translated to a tableau (or deduced that it is unsatisfiable) by representing the xor-constraints in matrix-like representation as above and translating it to reduced row echelon form by applying Gauss-Jordan elimination. Now we present the most important properties of tableaus with examples. Tableau operations are presented in more detail in [IV]. *All the implied equivalences* can be read from a tableau. Consider the xor-constraint conjunction

$$\phi_{\text{xor}} = (a \oplus b \oplus c \equiv \top) \wedge (b \oplus c \oplus d \equiv \bot) \wedge (c \oplus d \oplus e \equiv \top)$$

from which we can obtain the tableau

$$\begin{array}{lllll} a & := & d & & \oplus\top \\ b & := & & e & \oplus\top \\ c & := & d\oplus & e & \oplus\top \end{array}$$

where the first two equations with exactly one non-basic variable are the implied equivalences of ϕ_{xor}.

Xor-assumptions are handled separately by IGJ, meaning that assigned variables are not eliminated from the tableau. Instead, an *assigned tableau* $\langle \mathcal{E}, \tau \rangle$ has an associated (typically partial) truth assignment τ which must be consistent with $\mathcal{E}$. To effectively detect xor-implied literals, the tableau is modified on demand in such a way that xor-assumptions involve only only non-basic variables and xor-implied literals involve only basic variables. Consider the tableau

$$\begin{array}{lllll} a & := & d & & \oplus\top \\ b & := & & e & \oplus\top \\ c & := & d\oplus & e & \oplus\top \end{array}$$

and suppose that the variable d is assigned to $\top$. By the first equation, we can deduce the xor-implied literal $a \equiv \bot$. To handle an xor-assumption involving a basic variable, e.g. in the tableau above, $b \equiv \bot$, the tableau must modified by swapping the basic variable with an unassigned non-basic variable, so the variable b is swapped with e, giving the equation $e := b \oplus \top$. The process of swapping a basic variable with an unassigned non-basic variable is called *pivoting*. To maintain reduced row echelon form, after the swap, all occurrences of the new basic variable must be eliminated from other equations by substituting them with the right hand side of the basic variable's equation, e.g. e is substituted with $b \oplus \top$ in the third equation, giving the tableau:

$$\begin{array}{lllll} a & := & d & & \oplus\top \\ e & := & & b & \oplus\top \\ c & := & d\oplus & b & \oplus\top \end{array}$$

Now both d and b are assigned, so using the second and the third equations we can deduce the xor-implied literals $e \equiv \top$ and $c \equiv \bot$.

Implying clauses for xor-implied literals can be obtained directly using the relevant equation. Consider the assigned tableau:

$$\left\langle \left\{ \begin{array}{lllll} a & := & d & & \oplus\top \\ e & := & & b & \oplus\top \\ c & := & d\oplus & b & \oplus\top \end{array} \right\}, \{a \mapsto \bot, b \mapsto \bot, c \mapsto \bot, d \mapsto \top, e \mapsto \top\} \right\rangle$$

If an equation $x_i := x_{i,1} \oplus ... \oplus x_{i,k_i} \oplus p_i$ is used to deduce the xor-implied literal $x_i \equiv \tau(x_i)$, then the implying clause can be obtained by taking the disjunction $(x_i \equiv \tau(x_i)) \vee (x_{i,1} \not\equiv \tau(x_{i,1})) \cdots \vee (x_{i,k_i} \not\equiv \tau(x_{i,1}))$ where unary xor-constraints are interpreted as literals. For instance, the implying clause for the xor-implied literal $c \equiv \bot$ is $\neg c \vee \neg d \vee b$. Each equation in a tableau is a prime implicate of ϕ_{xor}, so implying clauses obtained from tableaus are minimal in the sense that they do not contain redundant literals. Implying clauses derived with the xor-deduction systems UP, Subst, and EC may contain redundant literals. Chapter 4 develops techniques for eliminating some of such redundant literals.

Implementation. Our implementation of the incremental Gauss-Jordan xor-reasoning module uses a dense matrix representation where one element in the matrix uses one bit of memory. The xor-reasoning module maintains two such matrices. In the first matrix the rows are consecutively in the memory, and in the second the columns are consecutively in the memory. The first matrix allows efficient implementation for row operations and the second matrix for efficient pivoting. To detect xor-implied literals, each row is associated with a counter tracking the number of unassigned variables. When this counter is one (or zero), an xor-implied literal (or a potential conflict) is available. Upon backtracking it suffices to restore the counters tracking unassigned variables.

Related work. The design of the xor-deduction system IGJ is inspired by the linear arithmetic solver by Dutertre and de Moura [2006]. The xor-deduction system IGJ is very similar to the one independently discovered by Han and Jiang [2012]. The main difference to our work is that i) we do not consider Craig interpolants (Craig [1957]), but ii) we can find all the implied equivalences. The implementation is also slightly different. The solver by Han and Jiang [2012] uses a form of 2-watched-literals technique adopted for bit arrays.

The solver march_eq by Heule and van Maaren [2004] applies Gaussian Elimination procedure as a preprocessing step to derive a minimal set of xor-constraints such that each xor-constraint in the set has a variable which does not occur in other xor-constraints. After this, some implied binary xor-constraints are extracted by taking linear combinations of pairs of xor-constraints until no new binary xor-constraints are found.

Katsirelos and Simon [2012] use Polynomial Calculus with Resolution (Alekhnovich *et al.* [2002]), PCR, to treat the CNF-xor formula as a whole. The proof system PCR generalizes both resolution and Gaussian elimination, so in theory it is more powerful than DPLL(XOR) with the xor-reasoning module IGJ. However, it is not obvious that such a system can be implemented efficiently enough to benefit from the stronger proof system when solving real-world instances.

The solver cryptominisat by Soos *et al.* [2009]; Soos [2010] uses Gaussian elimination i) to find xor-implied literals, and ii) to detect xor-conflicts. As it was already observed, Gaussian elimination, as used in cryptominisat, does not necessarily find all xor-implied literals. The implementation uses two matrices with Gaussian elimination. In the first one, xor-constraints are simplified with xor-assumptions and xor-implied literals. In the second one, only row and pivot operations are performed. The first matrix allows to detect xor-implied literals and xor-conflicts while the second matrix is used to compute implying clauses. Both matrices are stored in a dense, bit-packed format except for one column which stores the truth constants for each row. The memory layout of the matrices is optimized for efficient memory cache usage. Variable-disjoint xor-constraint sets are placed in separate matrix pairs. Gaussian elimination is performed only when the number of decisions is smaller than the configurable cut-off depth, and it is restricted to the part of the matrix where assignments have taken place. Gaussian elimination can also be automatically turned off by heuristic cut-off function which tracks how often Gaussian elimination deduces xor-implied literals and xor-conflicts. The matrices are updated starting from the leftmost column that has been changed (the corresponding variable has been assigned). Upon restarts, the columns are ordered according to heuristic values representing how actively variables participate in conflicts to increase the likelihood that the variables corresponding to the leftmost columns are assigned first. In addition, Gaussian elimination is further restricted to rows on the lower-right corner of the matrices as this is enough to maintain row-echelon form.

The decision procedure Gauss-DPLL to solve formulas consisting of clauses and xor-constraints by Baumgartner and Massacci [2000] has two inference rules Gauss^- and Gauss^+ that correspond to taking the linear combination of two xor-constraints provided that they share at least one variable. The deductions by the xor-deduction system IGJ can be simulated by the inference rules Gauss^- and Gauss^+ [1].

3.5 Experimental Evaluation

Here we give an overview of how the xor-deduction systems UP, Subst, IGJ, and EC presented in this chapter compare to each other in practice. The aim is not to outperform the reference solver, unmodified minisat version 2.0 core by Eén and Sörensson [2003], but to examine how much the search space (the number of heuristic decisions[2]) can be reduced by incorporating xor-reasoning into a SAT solver. There are many possible xor-reasoning module integration strategies and some work better than others on a given instance and a given xor-deduction system. Here we have chosen the integration strategy (out of the ones we have implemented) that gives the highest reduction in the number of heuristic decisions on A5/1 and Trivium benchmarks. The chosen integration strategy is slightly different from the essential skeleton of DPLL(XOR) in Figure 2.1, which typically gives better propagation performance. Instead of giving all xor-assumptions to the xor-reasoning module at once, one xor-assumption is given at a time and after that all new xor-implied literals are processed.

Figures 3.10, 3.11, and 3.13 show the results of running each solver configuration on each benchmark instance for at most one hour on 20-core Intel E5-2680 v2 with 256 GB RAM per processor. Memory limit for one solver instance was set to 10 GB. The xor-deduction system IGJ has the lowest median number of decisions for A5/1, FEAL, SAT Competition, and Trivium benchmarks, but on DES, Grain, and Hitag2 there is hardly any difference to UP, which performs plain unit propagation. Some SAT Competition instances consist only of xor-constraints, so IGJ solves them

[1]The inference rules Gauss^- and Gauss^+ are revisited in Section 4.3.

[2]We could have alternatively chosen to report the number of conflicts instead of the number of decisions. For some (artificial) formulas, such as $\bigwedge_{i=1}^{1000000}(a_i \vee \neg a_i)$, there can be a major difference in how many decisions a SAT solver requires to solve it compared to the number of conflicts. However, we have verified that for the reported benchmark instances, the number of decisions correlates strongly with the number of conflicts.

without needing xor-assumptions. The xor-deduction system Subst solves more Grain instances than IGJ, but IGJ performs significantly better than Subst on A5/1, FEAL, and SAT Competition benchmarks. The xor-deduction system IGJ also solves more Trivium instances than Subst.

DES, Grain, and Hitag2 benchmark instances are not solved with significantly fewer decisions with any xor-deduction system. The xor-reasoning module integration strategy in Figure 2.1, that emphasizes the CNF-part more, works better for these benchmarks. The DES benchmark instances have small and simple xor-clusters, so great reduction in the number of decisions is not expected. The structure of the DES benchmark instances is discussed more in [III]. Grain and Hitag2 benchmarks have large xor-clusters and xor-deduction systems can deduce many xor-implied literals, so we expected to see some reduction in the number of decisions. It is left for future work to study these instances further and to understand why these xor-deduction systems do not work as expected.

The motivation behind EC was to have an equally strong but more efficiently implementable proof system to Subst. However, this is not reflected in the results as EC consistently requires more decisions to solve instances than Subst, although it does solve more FEAL instances than Subst. This may be explained by how EC computes implying clauses for xor-implied literals. The xor-deduction system EC computes implying clauses using furthest cuts, which seems to work the best for EC, but Subst uses first CNF-compatible cuts, which give better results for Subst. Also, EC uses expensive breadth-first search to minimize implying clauses, which incurs a significant overhead. The equivalence reason graph used in EC gives more freedom to choose how implying clauses are computed, so ideally it could be used to compute "better" implying clauses than what Subst computes. It remains open whether EC can be made to compute "better" implying clauses and whether it can be implemented more efficiently than Subst. Even if Subst remains practically more relevant than EC, the xor-deduction system EC has theoretical value in succinctly characterizing equivalence reasoning which is shown in Section 5.2 to have a close connection with structural properties of xor-constraint conjunctions.

A5/1 (640 instances)			
Solver	#	Decisions	Time (s)
IGJ	640	3099	1.9
minisat	626	37096	5.0
UP	613	34910	5.0
SUBST	597	3134	5.2
EC	548	5762	19.4

DES (51 instances)			
Solver	#	Decisions	Time (s)
UP	51	72062	12.2
IGJ	51	89991	19.2
SUBST	51	91826	17.3
minisat	51	97652	14.5
EC	51	249363	73.7

FEAL (84 instances)			
Solver	#	Decisions	Time (s)
minisat	84	811828	54.1
UP	84	1594754	145.7
IGJ	79	73113	92.8
SUBST	21	-	-
EC	21	-	-

Grain (357 instances)			
Solver	#	Decisions	Time (s)
minisat	323	265328	282.1
UP	313	261088	275.1
SUBST	222	127654	1759.7
IGJ	186	239424	3267.4
EC	165	-	-

Hitag2 (306 instances)			
Solver	#	Decisions	Time (s)
minisat	295	1100849	280.0
UP	282	1404806	488.1
SUBST	190	2631385	2219.3
IGJ	171	1556543	3098.2
EC	102	-	-

SAT (474 instances)			
Solver	#	Decisions	Time (s)
IGJ	295	1094153	1069.5
minisat	279	14857305	1401.7
UP	271	18892454	1613.3
SUBST	212	-	-
EC	130	-	-

Trivium (1020 instances)			
Solver	#	Decisions	Time (s)
minisat	902	9361	3.8
UP	893	10751	5.8
IGJ	873	2569	21.3
SUBST	826	2872	47.5
EC	735	3732	142.0

Figure 3.10. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the seven benchmark families

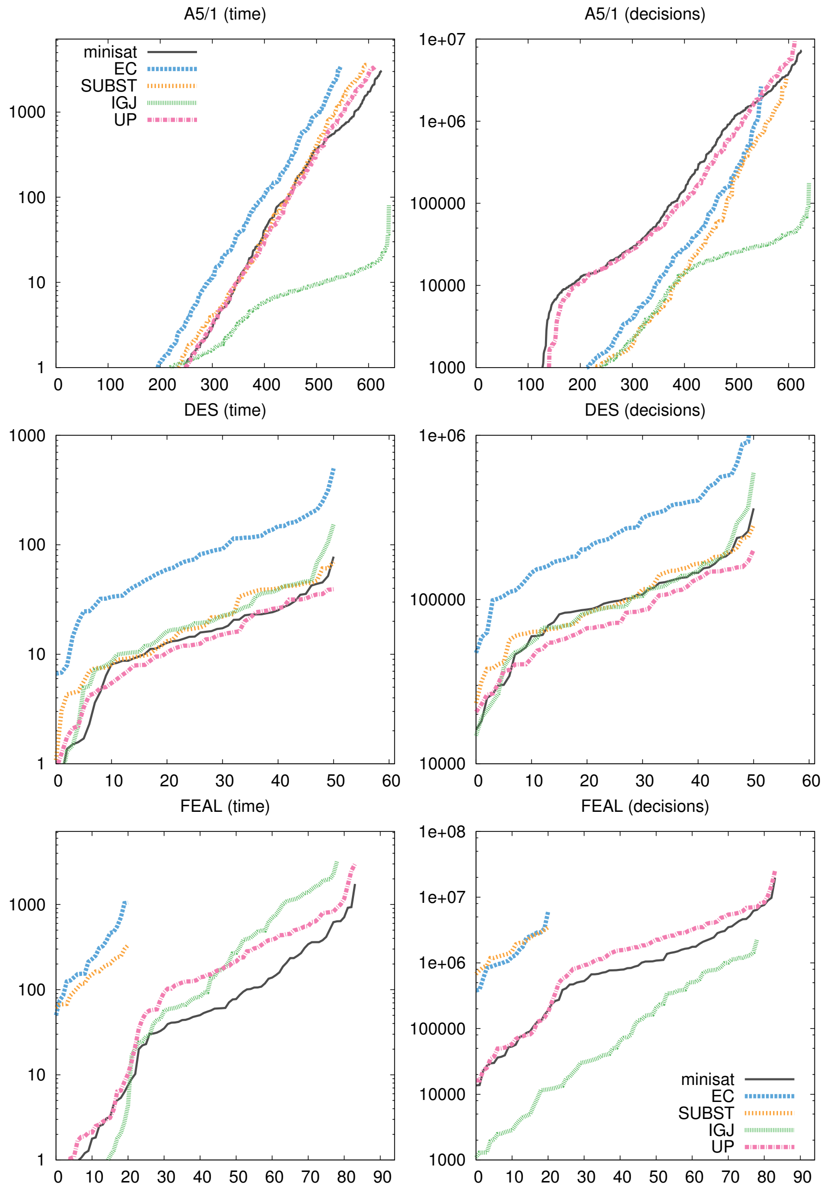


Figure 3.11. Solving time and number of decisions as functions of solved instances (part 1/3)

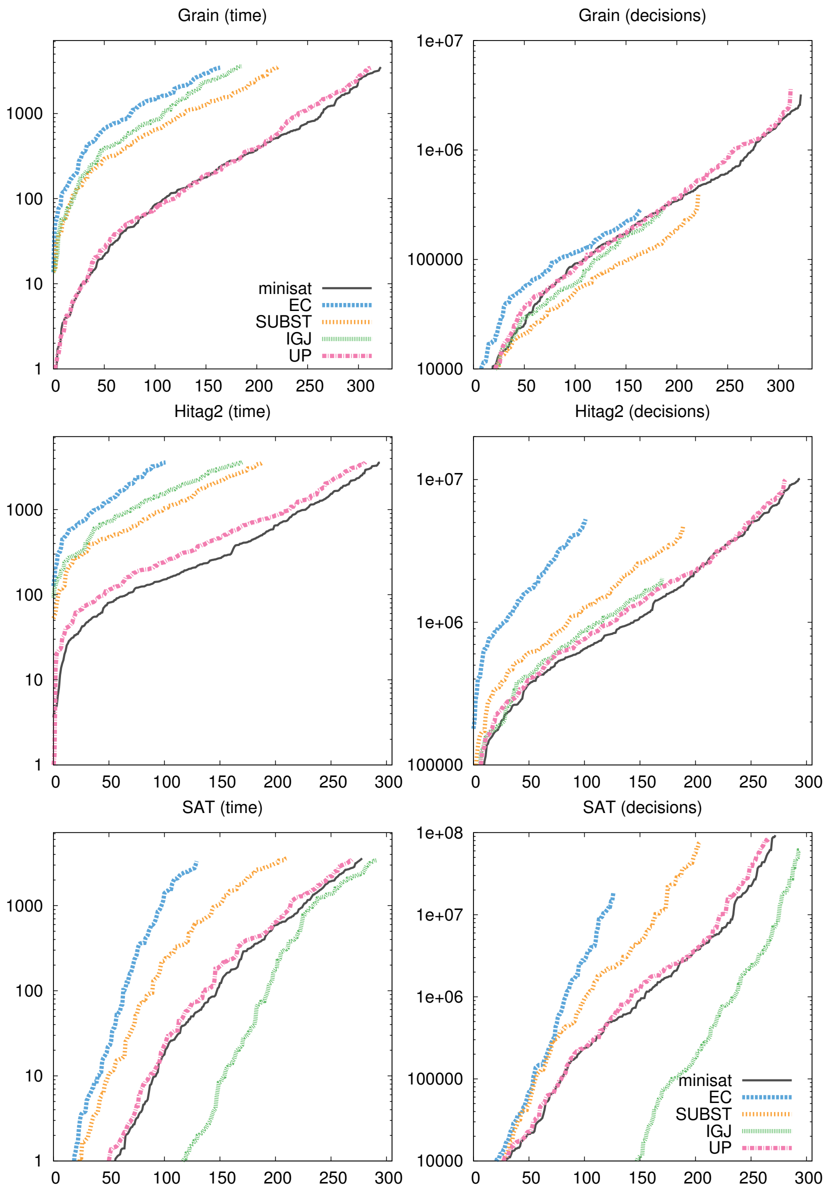


Figure 3.12. Solving time and number of decisions as functions of solved instances (part 2/3)

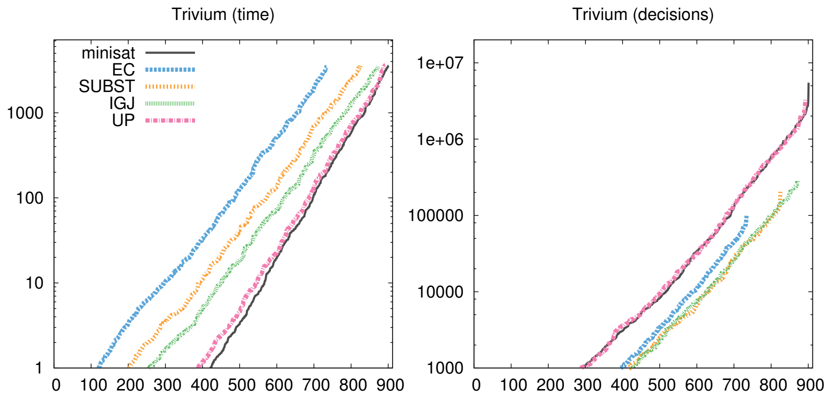


Figure 3.13. Solving time and number of decisions as functions of solved instances (part 3/3)

4. Parity Explanations

In this chapter, we develop new techniques to exploit structural properties of xor-constraints for xor-deduction systems based on xor-derivations, e.g. UP, Subst, and EC presented in Chapter 3. We introduce new techniques to explain why a literal was implied or why a conflict occurred in the xor-part; such explanations are needed by the CDCL part. The new core idea is to not see xor-level propagations as implications but as linear arithmetic equations. As a result, the new proposed *parity explanation* techniques can (i) provide smaller clausal explanations for the CDCL part, and also (ii) derive new xor-constraints that can then be learned in the xor-part. By learning new xor-constraints, we aim at, similarly to clause learning in CDCL solvers, enhancing the deduction capabilities of the xor-reasoning module. We introduce the new techniques using the very simple xor-deduction system UP, which allows only unit propagation on xor-constraints, and then show that the new parity explanation techniques also extend to more general xor-deduction systems, for instance to Subst, an xor-deduction system capable of equivalence reasoning in addition to unit propagation. We conclude the chapter by evaluating the effect of the proposed techniques experimentally.

4.1 From Implicative Explanations to Parity Explanations

As explained in Section 3.1, the CDCL part of the DPLL(XOR) framework requires an implying clause for each xor-implied literal. These can be computed by interpreting the $\oplus$-Unit^{+} and $\oplus$-Unit^{-} as implications. The UP xor-deduction system (and also the xor-deduction systems Subst and EC) use the inference rules in an "implicative way". For instance, the inference rule $\oplus$-Unit^{+} is implicitly interpreted as if the xor-constraints $(x \equiv \top)$ *and* C hold, *then* $C[x/\top]$ also holds.

Similarly, the implicative explanation for an xor-implied literal $\hat{l}$ labeling a non-input node v in a UP-derivation under a CNF-compatible cut W has been defined to be a conjunction $\mathit{Expl}(v, W)$ of literals with $\phi_{\text{xor}} \models \mathit{Expl}(v, W) \Rightarrow \hat{l}$ holding. We now propose an alternative method allowing us to compute a *parity explanation* $\mathit{Expl}_{\oplus}(v, W)$ that is an xor-constraint such that

$$\phi_{\text{xor}} \models \mathit{Expl}_{\oplus}(v, W) + \hat{l}$$

holds. The variables in $\mathit{Expl}_{\oplus}(v, W)$ will always be a subset of the variables in the implicative explanation $\mathit{Expl}(v, W)$ computed on the same cut.

The key observation for computing parity explanations is that the inference rules can in fact also be read as *equations* over xor-constraints under some provisos. As an example, the $\oplus$-Unit^{+} rule can be seen as the equation $(x \equiv \top) + C = C[x/\top]$ *provided that* (i) $x \in C$, and (ii) C is in the normal form. That is, the linear combination of the two premises equals the consequence xor-constraint of the rule. The provisos are easy to fulfill: (i) we have already assumed all xor-constraints to be in the normal form, and (ii) applying the rule when $x \notin C$ is redundant and can thus be disallowed. The reasoning is analogous for the $\oplus$-Unit^{-} rule and thus for UP rules we have the equations:

$$(x \equiv \top) + C \quad = \quad C[x/\top] \tag{4.1}$$

$$(x \equiv \bot) + C \quad = \quad C[x/\bot] \tag{4.2}$$

As all the UP-rules can be interpreted as equations of form “the linear combination of left-premise and right-premise equals the consequence”, we can expand any xor-constraint C in a node of a UP-derivation by iteratively replacing it with the left hand side of the corresponding equation. As a result, we will get a linear combination of xor-constraints that is logically equivalent to C; from this, we can eliminate the xor-constraints in ϕ_{xor} and get an xor-constraint D such that $\phi_{\text{xor}} \models D + C$. Formally, assume a UP-derivation $G = \langle V, E, L \rangle$ for $\phi_{\text{xor}} \wedge l_1 \wedge \ldots \wedge l_k$. For each non-input node v in G, and each cut $W = \langle V_{\text{a}}, V_{\text{b}} \rangle$ of G for v, the *parity explanation* of v under W is $\mathit{Expl}_{\oplus}(v, W) = f_W(v)$, where f_W is recursively defined as earlier for $\mathit{Expl}(v, W)$ except that the case “E4” is replaced by

E4 If u is a non-input node in V_{b}, then $f_W(u) = f_W(u_1) + f_W(u_2)$, where u_1 and u_2 are the source nodes of the two edges incoming to u.

We now illustrate parity explanations and show that they can be smaller (in the sense of containing fewer variables) than implicative explanations:

Example 5. *Consider again the* UP*-derivation given in Figure 3.2. Take the cut 4 first; we get* $\mathit{Expl}_{\oplus}(v_{12}, W) = c \oplus e \equiv \bot$*. Now we can verify that* $\phi_{\text{xor}} \models \mathit{Expl}_{\oplus}(v_{12}, W) + L(v_{12})$ *as* $\phi_{\text{xor}} \models (c \oplus e \equiv \bot) + (f \equiv \top)$*, i.e.* $c \oplus e \oplus f \equiv \top$ *is an xor-constraint in* ϕ_{xor}*. Observe that the implicative explanation* $c \wedge e$ *of* v_{12} *under the cut is just one conjunct in the disjunctive normal form* $(c \wedge e) \vee (\neg c \wedge \neg e)$ *of* $c \oplus e \equiv \bot$*.*

On the other hand, under the cut 2 we get $\mathit{Expl}_{\oplus}(v_{12}, W) = d$*. Now* $\phi_{\text{xor}} \models \mathit{Expl}_{\oplus}(v_{12}, W) + L(v_{12})$ *holds as* $\phi_{\text{xor}} \models (d \equiv \top) + (f \equiv \top)$*, i.e.* $d \oplus f \equiv \bot$ *is a linear combination of the xor-constraints in* ϕ_{xor}*. Note that the implicative explanation for* v_{12} *under the cut is* $(c \wedge d)$*, and no CNF-compatible cut for* v_{12} *gives the implicative explanation* (d) *for* v_{12}*.* ♣

We observe that $\text{vars}(\mathit{Expl}_{\oplus}(v, W)) \subseteq \text{vars}(\mathit{Expl}(v, W))$ by comparing the definitions of $\mathit{Expl}(v, W)$ and $\mathit{Expl}_{\oplus}(v, W)$. The correctness of $\mathit{Expl}_{\oplus}(v, W)$, formalized in the following theorem, can be established by induction and using Equations (4.1) and (4.2). Observe that the following theorem is a special case of Theorem 5 given later.

Theorem 3 (Thm. 1 of Laitinen *et al.* [2014b]). *Let* $G = \langle V, E, L \rangle$ *be a* UP*-derivation on* $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$*,* v *a node in it, and* $W = \langle V_{\text{a}}, V_{\text{b}} \rangle$ *a cut for* v*. The following fact holds:* $\phi_{\text{xor}} \models \mathit{Expl}_{\oplus}(v, W) + L(v)$*.*

Recall that the CNF-part solver requires an implying clause C for each xor-implied literal, forcing the value of the literal by unit propagation. A parity explanation can be used to get such implying clause by taking the implicative explanation as a basis and omitting the literals on variables not occurring in the parity explanation:

Theorem 4 (Thm. 2 of Laitinen *et al.* [2014b]). *Let* $G = \langle V, E, L \rangle$ *be a* UP*-derivation on* $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$*,* v *a node with* $L(v) = \hat{l}$ *in it, and* $W = \langle V_{\text{a}}, V_{\text{b}} \rangle$ *a CNF-compatible cut for* v*. Then* $\phi_{\text{xor}} \models (\bigwedge_{u \in S} L(u)) \Rightarrow \hat{l}$*, where* $S = \{u \in \text{reasons}(W) \mid \text{vars}(L(u)) \subseteq \text{vars}(\mathit{Expl}_{\oplus}(v, W))\}$*.*

Observing that only expressions of the type $f_W(u)$ occurring an odd number of times in the expression $f_W(v)$ remain in $\mathit{Expl}_{\oplus}(v, W)$, we can derive a more efficient graph traversal method to compute parity explanations. That is, when computing a parity explanation for a node, we traverse the derivation backwards from it in a breadth-first order. If we traverse a node u and notice that its traversal is requested because an even number of its successors have been traversed, then we don't need to traverse u further or include $L(u)$ in the explanation if u was on the "reason side" V_{a} of the cut.

Example 6. *Consider again the* UP*-derivation in Figure 3.2 and the CNF-compatible cut 1 for* v_{12}*. When we traverse the derivation backwards, we observe that the node* v_9 *has an even number of traversed successors; we thus don't traverse it (and consequently neither* v_8*,* v_5*,* v_4 *or* v_1*). On the other hand,* v_6 *has an odd number of traversed successors and it is included when computing* $\mathit{Expl}_{\oplus}(v_{12}, W)$*. Thus we get* $\mathit{Expl}_{\oplus}(v_{12}, W) = L(v_6) = (d)$ *and the implying clause for* f *is* $d \Rightarrow f$*, i.e.* $(\neg d \vee f)$*.* ♣

Although parity explanations can be computed quite fast using graph traversal as explained above, this can still be computationally prohibitive on "xor-intensive" instances because a single CNF-level conflict analysis may require that implying clauses for hundreds of xor-implied literals are computed. In our current implementation, we compute the closest CNF-compatible cut (for which parity explanations are very fast to compute but equal to implicative explanations and produce clausifications of single xor-constraints as implying clauses) for an xor-implied literal $\hat{l}$ when an explanation is needed in the regular conflict analysis. The computationally more expensive furthest cut is used if an explanation is needed again in the conflict-clause minimization phase of minisat. It is left for future work to develop a more efficient way to compute furthest cuts. Using dynamic programming to cache parity explanations for xor-implied literals may improve the performance significantly provided that the quadratic memory overhead is acceptable.

4.2 Learning Parity Explanations

As explained in the previous section, parity explanations can be used to derive implying clauses, which are required by the conflict analysis engine of the CDCL solver, that are shorter than those derived using the classic implicative explanations. In addition to this, parity explanations can be used to derive *new xor-constraints* that are logical consequences of ϕ_{xor}; these xor-constraints D can then be *learned*, meaning that ϕ_{xor} is extended to $\phi_{\text{xor}} \wedge D$, the goal being to increase the deduction power of the xor-deduction system. As an example, consider again Example 5 and recall that the parity explanation for v_{12} under the cut 2 is d. Now

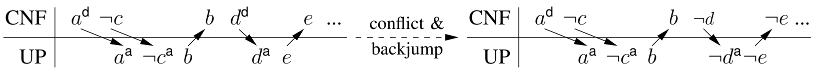

CNF
UP
a^d $\neg c$ b d^d e ...
a^a $\neg c^a$ b d^a e
conflict &
backjump
CNF
UP
a^d $\neg c$ b $\neg d$ $\neg e$ ...
a^a $\neg c^a$ b $\neg d^a$ $\neg e$

Figure 4.1. Communication between CNF-part and UP-module in a case when duplicate xor-constraints are learned; the d and a superscripts denote decision literals and xor-assumptions, respectively.

$\phi_{\text{xor}} \models (d \equiv \top) + (f \equiv \top)$, i.e. $\phi_{\text{xor}} \models (d \oplus f \equiv \bot)$, holds, and we can extend ϕ_{xor} to $\phi'_{\text{xor}} = \phi_{\text{xor}} \wedge (d \oplus f \equiv \bot)$ while preserving all the satisfying truth assignments. In fact, it is not possible to deduce $(f \equiv \top)$ from $\phi_{\text{xor}} \wedge (d \equiv \top)$ by using UP, but $(f \equiv \top)$ can be deduced from $\phi'_{\text{xor}} \wedge (d \equiv \top)$. Thus learning new xor-constraints derived from parity explanations can increase the deduction power of the UP xor-deduction system in a way similar to conflict-driven clause learning increasing the power of unit propagation in CDCL SAT solvers.

However, if all such derived xor-constraints are learned, it is possible to learn the same xor-constraint many times, as illustrated in the following example and Figure 4.1.

Example 7. *Let $\phi_{\text{xor}} = (a \oplus b \oplus c \equiv \bot) \wedge (b \oplus c \oplus d \oplus e \equiv \top) \wedge ...$ and assume that CNF-part solver gives its first decision level literals a and $\neg c$ as xor-assumptions to the* UP*-module; the module deduces b and returns it to the CNF solver. At the next decision level the CNF-part guesses d, gives it to* UP*-module, which deduces e, returns it to the CNF-part, and the CNF-part propagates it so that a conflict occurs. Now the xor-implied literal e is explained and a new xor-constraint $D = (a \oplus d \oplus e \equiv \bot)$ is learned in ϕ_{xor}. After this the CNF-part backtracks, implies $\neg d$ at the decision level 1, and gives it to the* UP*-module; the module can then deduce $\neg e$* without *using D. If $\neg e$ is now explained, the same "new" xor-constraint $(a \oplus d \oplus e \equiv \bot)$ can be derived.* ♣

The example illustrates a commonly occurring case in which a derived xor-constraint contains two or more literals on the latest decision level (e and d in the example); in such a case, the xor-constraint may already exist in ϕ_{xor}. A conservative approach to avoid learning the same xor-constraint twice, under the reasonable assumption that the CNF and xor-reasoning module parts saturate their propagations before new heuristic decisions are made, is to disregard derived xor-constraints that have two or more variables assigned on the latest decision level. If a learned xor-constraint for xor-implied literal $\hat{l}$ does not have other literals on the latest decision level, it can be used to infer $\hat{l}$ with fewer decision literals. Note that it may also happen that an implying clause for an xor-implied literal $\hat{l}$ does not contain any literals besides $\hat{l}$ on the latest decision level; the CNF-part may then compute a conflict clause that does not have any literals on the current decision level, which needs to be handled appropriately.

In order to avoid slowing down propagation in our implementation, we store and remove learned xor-constraints using a strategy adopted from minisat: the maximum number of learned xor-constraints is increased at each restart and the "least active" learned xor-constraints are removed when necessary. However, using the conservative approach to learning xor-constraints, the total number of learned xor-constraints rarely exceeds the number of original xor-constraints.

4.3 Generalizing Parity Explanations

So far in this chapter we have considered the xor-deduction system UP which is only capable of unit propagation. We can in fact extend the introduced concepts to more general inference systems and derivations.

Define an *xor-derivation* similarly to UP-derivation except that there is only one inference rule:

$$\oplus\text{-Gen} : \frac{D_1 \quad D_2}{D_1 + D_2}$$

where D_1 and D_2 are xor-constraints. The inference rule $\oplus$-Gen is a generalization of the rules Gauss^{-} and Gauss^{+} in Baumgartner and Massacci [2000]. Now Theorems 3 and 4 can be shown to hold for such derivations as well. The following theorem establishes the correctness of parity explanations on (general) xor-derivations using the inference rule $\oplus$-Gen:

Theorem 5. *Let $G = \langle V, E, L \rangle$ be an xor-derivation on $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$, v a node in it, and $W = \langle V_{\text{a}}, V_{\text{b}} \rangle$ a cut for v. The following fact holds: $\phi_{\text{xor}} \models \mathit{Expl}_{\oplus}(v, W) + L(v)$.*

Proof. We show that $\phi_{\text{xor}} \models (f_W(u) + L(u))$ for each node u in G by induction on the structure of the derivation G.

If u is an input node with $L(u) \in \phi_{\text{xor}}$, then $f_W(u) = \bot \equiv \bot$ and $\phi_{\text{xor}} \models (\bot \equiv \bot) + L(u)$ as $L(u)$ is a conjunct in ϕ_{xor}.

If u is an input node with $L(u) \in \{l_1, ..., l_k\}$, then $f_W(u) = L(u)$ and $\phi_{\text{xor}} \models L(u) + L(u)$ holds trivially.

If u is a non-input node in V_{a}, then $f_W(u) = L(u)$ and $\phi_{\text{xor}} \models (L(u) + L(u))$ holds trivially.

If u is a non-input node in V_b with two incoming edges from nodes u_1 and u_2 respectively, then $f_W(u) = f_W(u_1) + f_W(u_2)$. Because $L(u)$ is obtained from $L(u_1)$ and $L(u_2)$ by applying $\oplus$-Gen, we have established that $\phi_\mathrm{xor} \models (L(u_1) + L(u_2)) + L(u)$. By the induction hypothesis the equations $\phi_\mathrm{xor} \models (f_W(u_1)+L(u_1))$ and $\phi_\mathrm{xor} \models (f_W(u_2)+L(u_2))$ hold. By taking the linear combination of these equations we get the equation $\phi_\mathrm{xor} \models (f_W(u_1) + f_W(u_2)) + (L(u_1) + L(u_2))$. We now take the linear combination of this and the previously established equation $\phi_\mathrm{xor} \models (L(u_1)+L(u_2))+L(u)$ resulting in $\phi_\mathrm{xor} \models (f_W(u_1)+f_W(u_2))+L(u)$ and substituting $(f_W(u_1)+f_W(u_2))$ with $f_W(u)$ using the equation $f_W(u) = f_W(u_1) + f_W(u_2)$ we get the equation $\phi_\mathrm{xor} \models (f_W(u) + L(u))$.

We have established for each u in G that $\phi_\mathrm{xor} \models (f_W(u) + L(u))$ holds. It follows that $\phi_\mathrm{xor} \models \mathit{Expl}_\oplus(v, W) + L(v)$. □

A parity explanation on a (general) xor-derivation can be used to get an implying clause by taking the implicative explanation as a basis and omitting the literals on variables not occurring in the parity explanation:

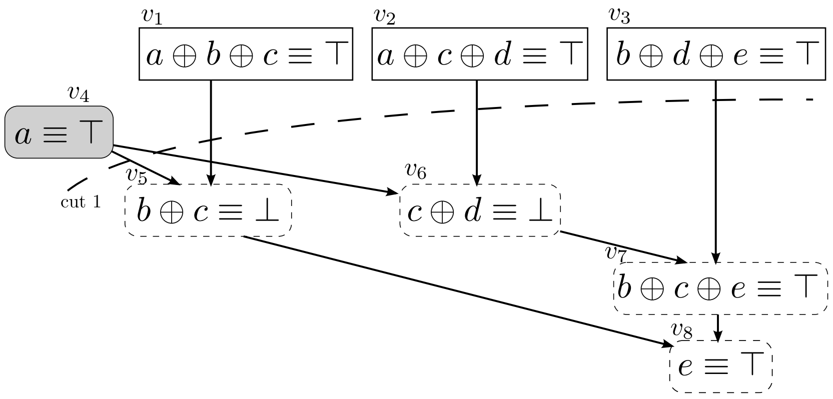
Theorem 6. *Let $G = \langle V, E, L\rangle$ be an xor-derivation on $\phi_\mathrm{xor} \wedge l_1 \wedge \cdots \wedge l_k$, v a node with $L(v) = \hat{l}$ in it, and $W = \langle V_\mathrm{a}, V_\mathrm{b}\rangle$ a CNF-compatible cut for v. Then $\phi_\mathrm{xor} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \hat{l}$, where $S = \{u \in \mathrm{reasons}(W) \mid \mathrm{vars}(L(u)) \subseteq \mathrm{vars}(\mathit{Expl}_\oplus(v, W))\}$.*

Proof. If ϕ_xor is unsatisfiable, then $\phi_\mathrm{xor} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \hat{l}$ holds trivially.

Assume that ϕ_xor is satisfiable. As $\mathrm{vars}(\bigwedge_{u\in S} L(u)) = \mathrm{vars}(\mathit{Expl}_\oplus(v, W))$ and $\phi_\mathrm{xor} \models \mathit{Expl}_\oplus(v, W) + \hat{l}$, it must be that either $\phi_\mathrm{xor} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \hat{l}$ or $\phi_\mathrm{xor} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \neg\hat{l}$ holds (both cannot hold because then ϕ_xor would be unsatisfiable). If $\phi_\mathrm{xor} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \neg\hat{l}$ holds, then, as $S \subseteq \mathrm{reasons}(W)$ holds, $\phi_\mathrm{xor} \models (\bigwedge_{u\in \mathrm{reasons}(W)} L(u)) \Rightarrow \neg\hat{l}$ would also hold. Combined with the property $\phi_\mathrm{xor} \models (\bigwedge_{u\in \mathrm{reasons}(W)} L(u)) \Rightarrow \hat{l}$ of implicative explanations, this means that ϕ_xor is unsatisfiable, contradicting our assumption. Therefore, $\phi_\mathrm{xor} \models (\bigwedge_{u\in S} L(u)) \Rightarrow \hat{l}$ must hold. □

We have seen in Section 3.1 that the inference rules $\oplus$-Unit$^+$ and $\oplus$-Unit$^-$ of UP xor-deduction system are a special case of $\oplus$-Gen. As another concrete example of xor-reasoning module implementing a sub-class of $\oplus$-Gen, consider the Subst module presented in Section 3.2. In addition to the unit propagation rules of UP in Figure 3.1, it has two inference rules allowing equivalence reasoning:

$$\oplus\text{-Eqv}^+ : \frac{x \oplus y \equiv \bot \quad C}{C\,[x/y]} \qquad\qquad \oplus\text{-Eqv}^- : \frac{x \oplus y \equiv \top \quad C}{C\,[x/(y \oplus \top)]}$$



v_1
$a \oplus b \oplus c \equiv \top$
v_2
$a \oplus c \oplus d \equiv \top$
v_3
$b \oplus d \oplus e \equiv \top$
v_4
$a \equiv \top$
cut 1
v_5
$b \oplus c \equiv \bot$
v_6
$c \oplus d \equiv \bot$
v_7
$b \oplus c \oplus e \equiv \top$
v_8
$e \equiv \top$

Figure 4.2. A Subst-derivation

where the symbols x and y are variables while C is an xor-constraint in the normal form with an occurrence of x. Note that these Subst rules are indeed instances of the more general inference rule $\oplus$-Gen. For instance, given two xor-constraints $C_1 = (c \oplus d \equiv \bot)$ and $C_2 = (b \oplus d \oplus e \equiv \top)$, Subst can produce the xor-constraint $C_2\,[d/c] = (b \oplus c \oplus e \equiv \top)$ which is also inferred by $\oplus$-Gen: $(C_1 + C_2) = (c \oplus d \oplus b \oplus d \oplus e \equiv \bot \oplus \top) = (b \oplus c \oplus e \equiv \top)$.

As an example, Figure 4.2 shows a Subst-derivation on $\phi_{\text{xor}} \wedge (a \equiv \top)$, where $\phi_{\text{xor}} = (a \oplus b \oplus c \equiv \top) \wedge (a \oplus c \oplus d \equiv \top) \wedge (b \oplus d \oplus e \equiv \top)$. The literal e is Subst-derivable on $\phi_{\text{xor}} \wedge (a)$; the xor-reasoning module returns e as an xor-implied literal on ϕ_{xor} after a is given as an xor-assumption. The CNF-compatible cut 1 for the literal e gives the implicative explanation (a) and thus the implying clause $(\neg a \vee e)$ for e. Parity explanations are defined for Subst in the same way as for UP; the parity explanation for the literal e in the figure is $\top$ and thus the implying clause for e is (e).

Parity explanations can also be defined in a similar way for the xor-deduction system EC that is based on equivalence class manipulation. For instance, consider the equivalence reason graph in Figure 3.8. By taking the union of reasons for the edges of the cyclic path with an odd number of disequality edges $b \overset{i \equiv \top}{\not\text{---}} g \overset{f \equiv \top}{\text{---}} h \overset{i \equiv \top}{\text{---}} e \overset{d \equiv \top}{\text{---}} c \overset{a \equiv \top}{\text{---}} b$, we get an implicative explanation $(\neg a \vee \neg d \vee \neg i \vee \neg f)$ for the xor-conflict. By omitting those edge labels which occur an even number of times, we get a parity explanation $(a \oplus d \oplus f \equiv \top)$ for the xor-conflict.

4.4 Experimental Evaluation

This section is a continuation to Section 3.5 that studies how much the search space (the number of heuristic decisions) can be reduced by integrating the xor-deduction systems UP, Subst, and IGJ to minisat version 2.0 core by Eén and Sörensson [2003]. Now we evaluate how the xor-deduction system UP enhanced with parity explanations performs on the same benchmarks. The reference solver configuration here is UP+fcut, where fcut stands for furthest cut. It computes implicative explanations using furthest cuts selectively when an implying clause is needed in the conflict-clause minimization phase (see Section 4.1 for details). The solver configuration UP+pexp is otherwise identical to UP+fcut but computes shorter implying clauses using parity explanations when furthest cuts are used. The solver configuration SUBST+p is otherwise identical to SUBST but it computes implying clauses using parity explanations always. The solver configuration UP+learn adds xor-constraint learning (see Section 4.2) to UP+pexp. The solvers minisat, SUBST, and IGJ are included for comparison.

Figures 4.3, 4.4, 4.5, and 4.6 show the results of running each solver configuration on each benchmark instance for at most one hour on 20-core Intel E5-2680 v2 with 256 GB RAM per processor. Memory limit for one solver instance was set to 10 GB. Using parity explanations in the conflict-clause minimization phase but without learning them does not give noticeable advantage on most of these benchmarks, that is, UP+fcut and UP+pexp perform similarly. The notable exception is the FEAL benchmark family whose instances are solved faster without learning parity explanations. Only 21 instances are solved by SUBST, but SUBST+p solves all instances with lowest median time and lowest median number of decisions. However, UP+pexp still performs similarly to UP+fcut. Adding parity explanations as learned xor-constraints reduces the number of decisions significantly for the benchmarks A5/1, SAT, and Trivium and this is reflected also in the solving time of UP+learn. The Grain benchmarks are solved with fewer decisions, but the computational overhead is too high, so UP+learn manages to solve fewer Grain instances than UP+pexp.

The solver configuration IGJ clearly outperforms UP+learn when solving A5/1 and FEAL benchmark instances, but solves fewer instances of other benchmark families. Section 7.3 compares the xor-deduction system UP with parity explanations to Gauss-Jordan elimination on a theoretical level. Chapter 6 presents techniques to improve solving performance for IGJ i) by decomposing the xor-part into distinct xor-constraint conjunctions that can be handled in separate (dense) matrices and ii) by eliminating variables that have occurrences only in the xor-part.

The solver SUBST+p solves more instances than SUBST on most benchmark families, and the difference is noticeable on A5/1, FEAL, Hitag2, and SAT benchmark families. Only Grain and Trivium benchmark families are solved faster with SUBST. Using parity explanations in the solver SUBST+p does not incur a significant overhead, so it is not clear why SUBST+p performs worse than SUBST on the Grain and Trivium benchmarks.

A5/1 (640 instances)				DES (51 instances)			
Solver	#	Decisions	Time (s)	Solver	#	Decisions	Time (s)
IGJ	640	3099	1.9	IGJ	51	89991	19.2
UP+learn	639	20651	4.2	UP+learn	51	90122	16.2
minisat	626	37096	5.0	SUBST+p	51	90439	17.0
UP+fcut	617	30036	5.6	SUBST	51	91826	17.3
UP+pexp	611	32745	5.6	UP+pexp	51	91832	17.6
SUBST+p	603	5289	7.6	UP+fcut	51	95881	16.6
SUBST	597	3134	5.2	minisat	51	97652	14.5
FEAL (84 instances)				**Grain (357 instances)**			
Solver	#	Decisions	Time (s)	Solver	#	Decisions	Time (s)
SUBST+p	84	200141	25.6	minisat	323	265328	282.1
minisat	84	811828	54.1	UP+pexp	315	271660	320.8
UP+fcut	84	1426547	150.8	UP+fcut	306	244842	267.9
UP+pexp	83	1131949	109.2	UP+learn	290	153123	420.0
IGJ	79	73113	92.8	SUBST	222	127654	1759.7
UP+learn	35	-	-	SUBST+p	198	247511	2773.1
SUBST	21	-	-	IGJ	186	239424	3267.4
Hitag2 (306 instances)				**SAT (474 instances)**			
Solver	#	Decisions	Time (s)	Solver	#	Decisions	Time (s)
minisat	295	1100849	280.0	UP+learn	301	1930731	428.3
UP+pexp	284	1358081	459.0	IGJ	295	1094153	1069.5
UP+learn	282	1028143	357.2	minisat	279	14857305	1401.7
UP+fcut	281	1247305	434.0	UP+fcut	269	22568178	2231.6
SUBST+p	247	1661239	1068.4	SUBST+p	267	14226864	2207.2
SUBST	190	2631385	2219.3	UP+pexp	267	28097432	2007.9
IGJ	171	1556543	3098.2	SUBST	212	-	-

Trivium (1020 instances)			
Solver	#	Decisions	Time (s)
UP+learn	907	6739	8.9
minisat	902	9361	3.8
UP+fcut	881	9718	5.2
UP+pexp	880	9442	5.1
IGJ	873	2569	21.3
SUBST	826	2872	47.5
SUBST+p	803	2985	60.4

Figure 4.3. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the seven benchmark families

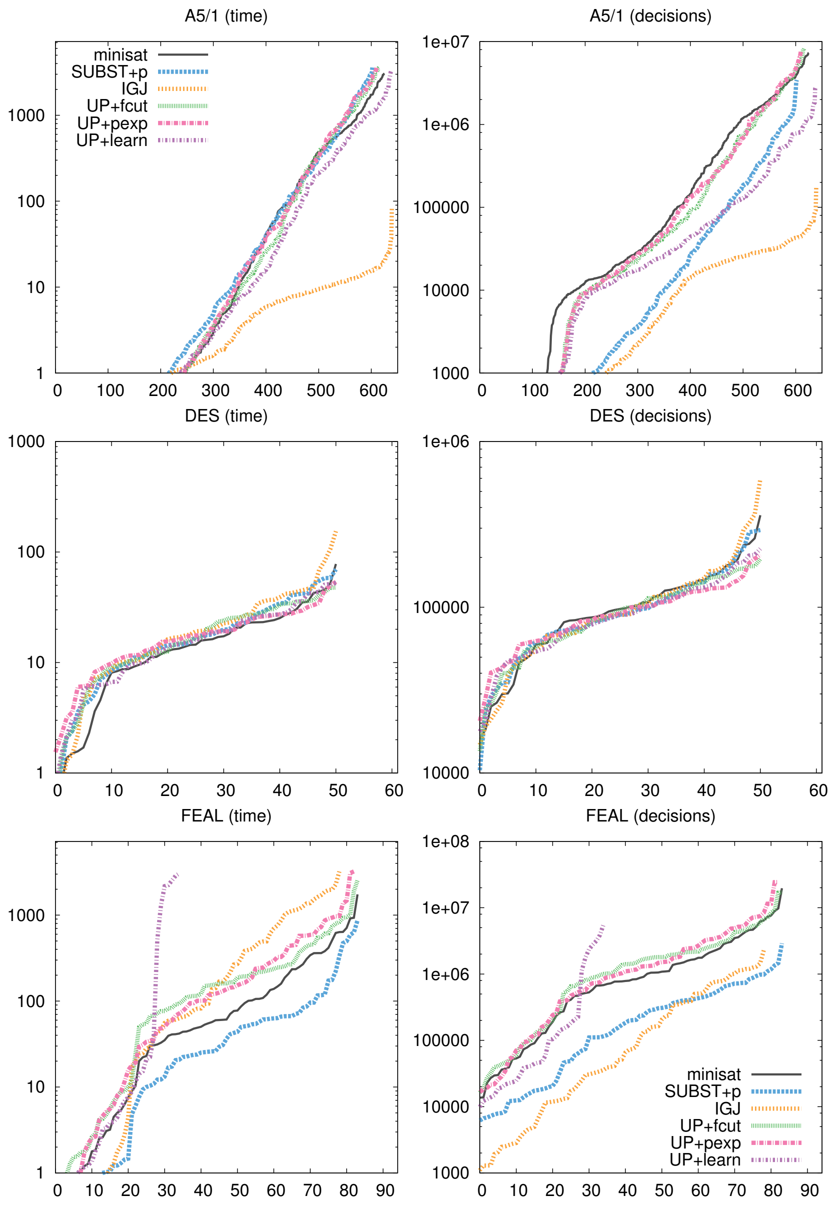


Figure 4.4. Solving time and number of decisions as functions of solved instances (part 1/3)

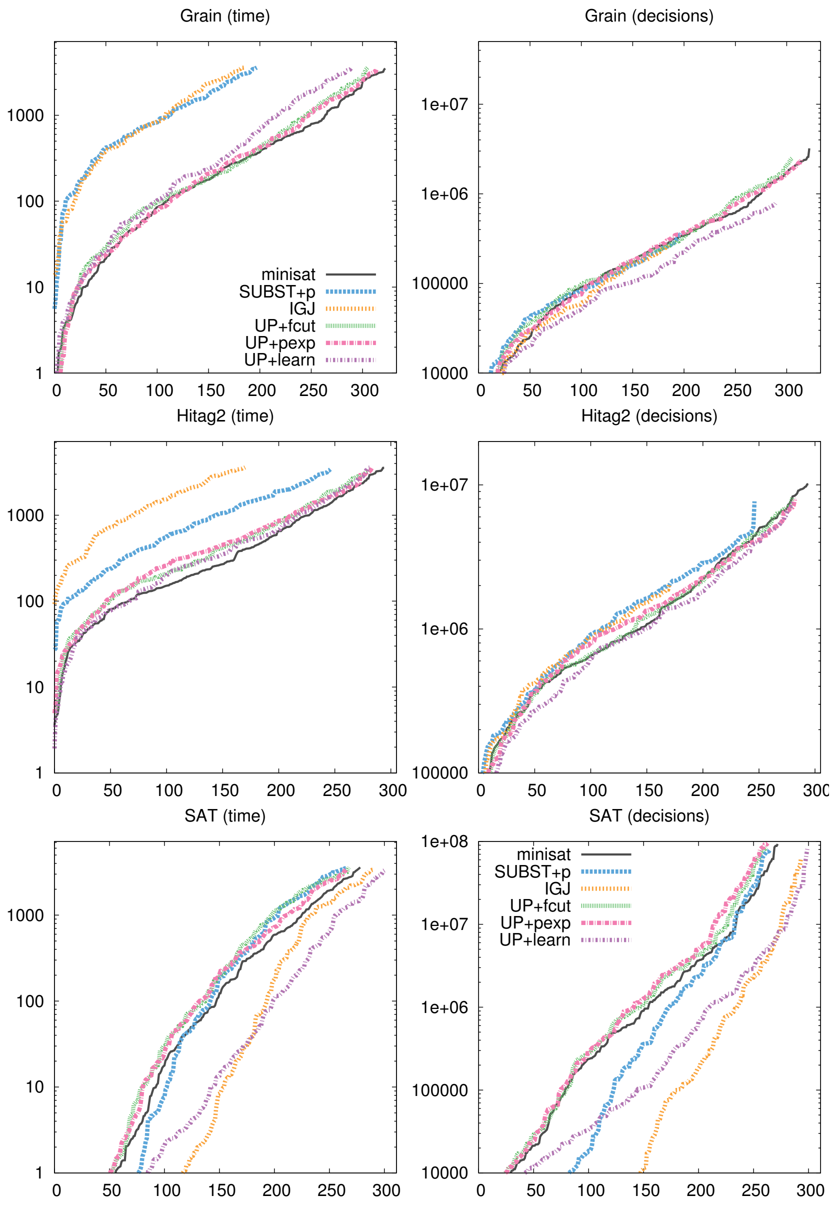


Figure 4.5. Solving time and number of decisions as functions of solved instances (part 2/3)

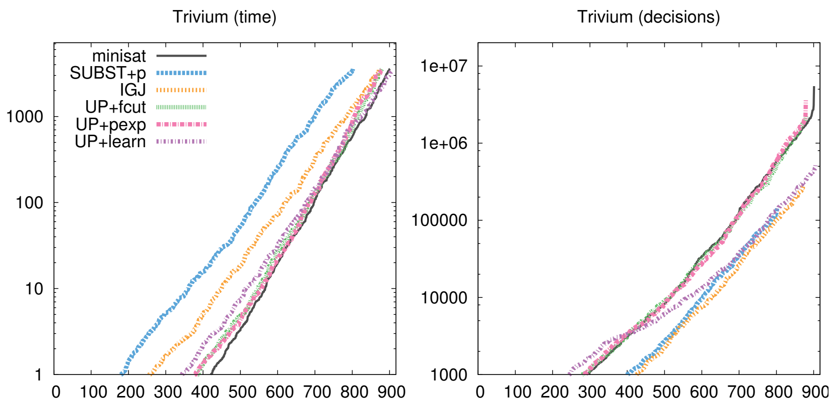


Figure 4.6. Solving time and number of decisions as functions of solved instances (part 3/3)

5. Classifying Parity Constraints

So far in this thesis, we have developed different xor-deduction systems to detect unsatisfiability of a set of parity constraints and deducing xor-implied literals. The presented xor-deduction systems have different trade-offs between proof system strength and computational complexity. Using stronger parity reasoning may prune the search space efficiently, but incurs some computational overhead. The "structure" of parity constraints ultimately dictates whether the search space can be pruned by stronger parity reasoning. In an optimal setting, an actual implementation of an xor-deduction system would carry out only the necessary propagation steps to deduce deducible xor-implied literals or to detect unsatisfiability, but in practice it can be challenging to engineer such an implementation.

In this chapter, we develop efficient approximating tests to decide if unit propagation or equivalence reasoning is enough to achieve full propagation in a given conjunction of parity constraints. The part of a set of parity constraints for which stronger parity reasoning cannot detect more xor-implied literals can then be handled by unit propagation leaving a smaller set of parity constraint for stronger xor-deduction systems. By analyzing the constraint graph primed by a set of parity constraints, we can characterize equivalence reasoning using the cycles in the graph. The presence of cycles in the graph indicates that equivalence reasoning might be useful, so absence of such cycles indicates that unit propagation is enough for so called "tree-like" parts. If the constraint graph has cycles, equivalence reasoning might be useful, but it does not give any indication of whether it is enough to deduce all xor-implied literals. We develop an approximating test to detect whether the cycles in the graph belong to a restricted class for which equivalence reasoning can detect all xor-implied literals.

This kind of syntactic analysis bears a resemblance to a number of techniques including (i) detecting polynomially solvable classes of CNF consisting of Horn-clauses and binary clauses (Fourdrinoy *et al.* [2007]), (ii) detecting and exploiting symmetries in CNF (Darga *et al.* [2004]), (iii) polynomial algorithms for solving “tree-like” constraint satisfaction problems (chapter 9 in Dechter [2003]), and (iv) recovering and exploiting other “hidden structure” in CNF to recognize and solve instances within some polynomial bound (Samer and Szeider [2009]).

5.1 When Unit Propagation Deduces All Xor-Implied Literals

We first study the problem of deciding, given an xor-constraint conjunction, whether unit propagation can always deduce all xor-implied literals. We use the xor-deduction system UP to define when unit propagation can always deduce all xor-implied literals:

Definition 2. *A conjunction ϕ_{xor} of xor-constraints is* UP-deducible *if for all $l_1, ..., l_k, \hat{l} \in \text{lits}(\phi_{\text{xor}})$ the following facts hold: (i) if $\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_k$ is unsatisfiable, then $\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_k \vdash_{\mathsf{UP}} \bot$, and (ii) $\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_k \models \hat{l}$ implies $\phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_k \vdash_{\mathsf{UP}} \hat{l}$ otherwise.*

We analyze what xor-deduction systems can deduce by using a graph primed by a conjunction of xor-constraints.

Definition 3. *A constraint graph of an xor-constraint conjunction ϕ_{xor} is a bipartite graph $\langle V, E\rangle$ where each xor-constraint in ϕ_{xor} and each variable in $\text{vars}(\phi_{\text{xor}})$ has a corresponding node in V and there is an edge in E between a variable node and an xor-constraint node if the variable has an occurrence in the xor-constraint.*

Although we do not know of an easy way of detecting whether a given conjunction of xor-constraints is UP-deducible, we show that “tree-like” xor-constraint conjunctions are UP-deducible.

Definition 4. *A conjunction ϕ_{xor} is tree-like if its constraint graph is a tree or a union of disjoint trees.*

Example 8. *The conjunction $(a \oplus b \oplus c \equiv \top) \wedge (b \oplus d \oplus e \equiv \top) \wedge (c \oplus f \oplus g \oplus \equiv \bot)$ is tree-like; its constraint graph is given in Figure 5.1(a). On the other hand, the conjunction $(a \oplus b \oplus c \equiv \top) \wedge (a \oplus d \oplus e \equiv \top) \wedge (c \oplus d \oplus f \equiv \top) \wedge (b \oplus e \oplus f \equiv \top)$, illustrated in Figure 5.1(b), is not tree-like.*

Theorem 7 (Thm. 1 of Laitinen *et al.* [2014a])**.** *If a conjunction of xor-constraints ϕ_{xor} is tree-like, then it is* UP*-deducible*

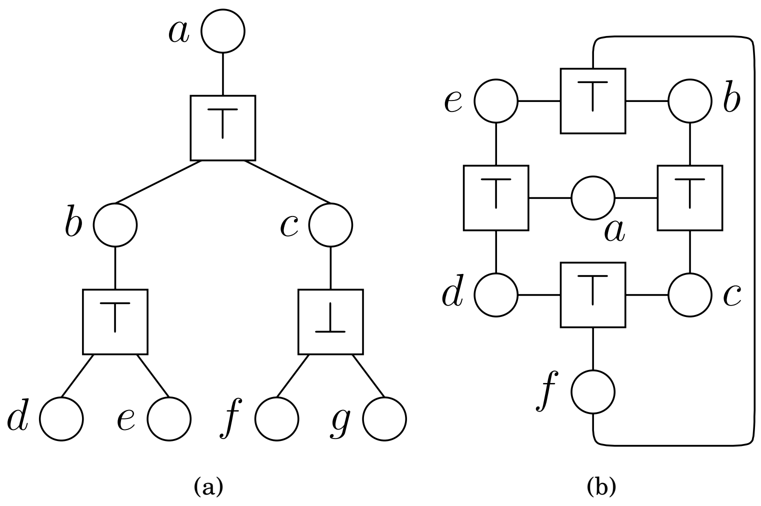


Figure 5.1. Two constraint graphs

Note that not all UP-deducible xor-constraints are tree-like. For instance, $(a \oplus b \equiv \top) \wedge (b \oplus c \equiv \top) \wedge (c \oplus a \equiv \bot)$ is satisfiable and UP-deducible but not tree-like. No binary xor-constraints are needed to establish the same, e.g., $(a \oplus b \oplus c \equiv \top) \wedge (a \oplus d \oplus e \equiv \top) \wedge (c \oplus d \oplus f \equiv \top) \wedge (b \oplus e \oplus f \equiv \top)$ considered in Example 8 is satisfiable and UP-deducible but not tree-like.

5.1.1 Experimental Evaluation

We have studied the SAT Competition benchmark instances (see Section 2.6) and found out that out of these 474 instances, 61 are tree-like. As shown in the earlier example, there exists UP-deducible instances that are not tree-like. To test whether any of the 413 non-tree-like instances we found are UP-deducible, we generated randomized saturated UP-derivations from those instances and then tested with the IGJ xor-deduction system (Gauss-Jordan elimination) whether unit propagation could detect all xor-implied literals.

The randomized testing could prove for all except one of the 413 non-tree-like instances that they are not UP-deducible. In these instances tree-like classification seems to approximate quite well UP-deducibility. Detailed results are shown in Figure 5.2(a). The columns "probably Subst" and "cycle-partitionable" are explained later. In the non-tree-like instances, on average 11% of the xor-constraints are tree-like.

The instances for which unit propagation detects all xor-implied literals do not benefit from stronger parity reasoning. To confirm this, we ran

	SAT Competition			
	2005	2007	2009	2011
instances	857	376	573	1200
with xors	123	100	140	111
probably UP	19	10	18	15
tree-like	19	9	18	15
probably Subst	20	21	52	40
cycle-partitionable	20	13	24	40

Figure 5.2. Instance classification

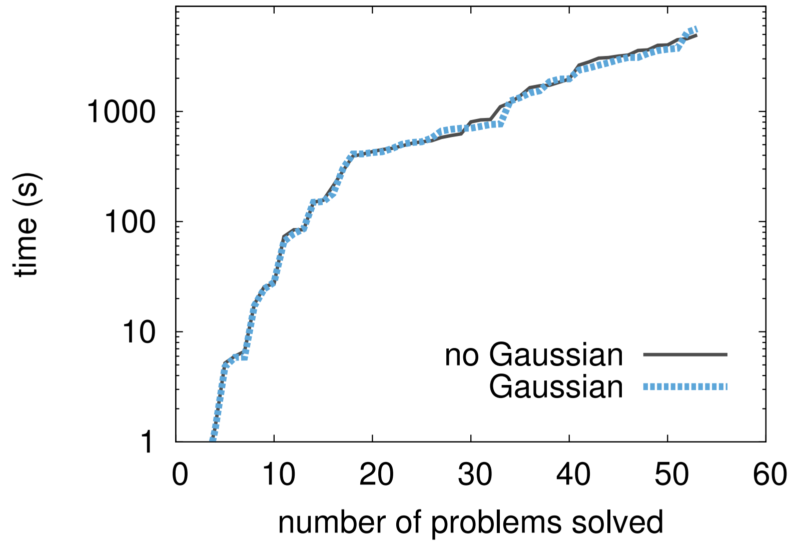


Figure 5.3. cryptominisat run-times on tree-like instances

cryptominisat 2.9.2 on the 61 aforementioned tree-like instances in two ways on 12-core Intel X5650 with 48GB memory (2 GB memory limit per task). In the first run, parity reasoning was disabled. In the second run, full Gaussian elimination was used. The results in Figure 5.3 confirm that for these tree-like instances it is better to use plain unit propagation instead of Gaussian elimination.

Our other benchmark families (see Chapter 2.6) exhibit a more homogeneous structure, so the average proportions of tree-like xor-constraints shown in Figure 5.4 are descriptive.

Benchmark family	Tree-like	All	Tree-like ratio
A5/1	716.5	1332.5	0.54
FEAL	0.0	3933.7	0.00
Grain	661.8	664.7	0.10
DES	198.4	542.3	0.37
Hitag2	1060.5	3909.5	0.27
Trivium	0.0	8167.4	0.00

Figure 5.4. Average number of tree-like xor-constraints in other benchmark families

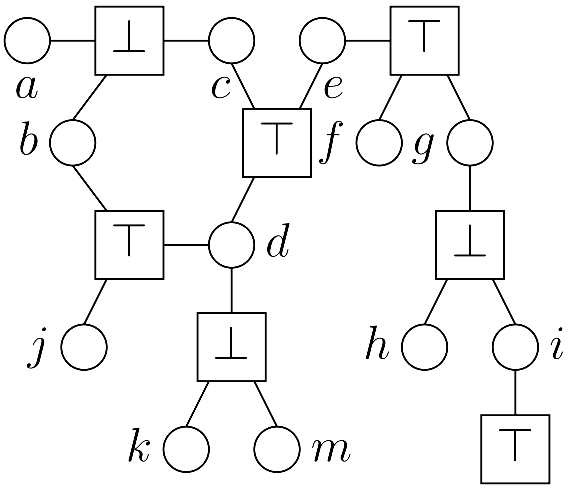


Figure 5.5. A constraint graph

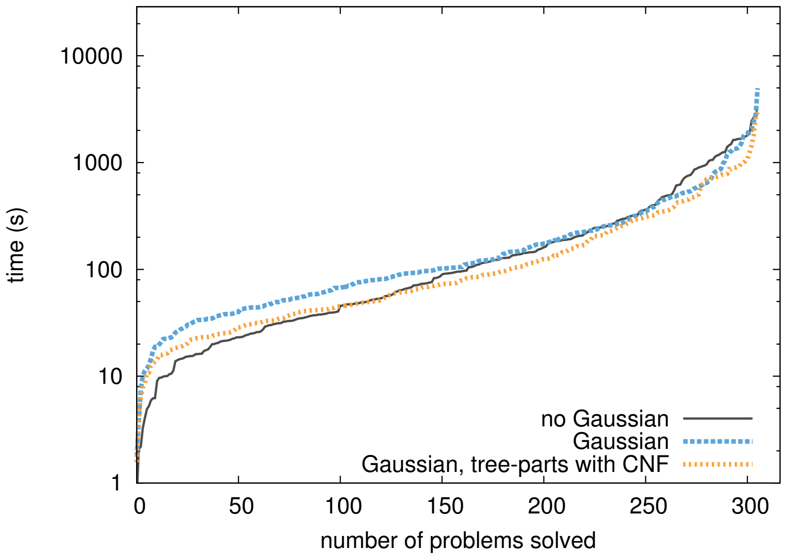


Figure 5.6. Run-times of cryptominisat on Hitag2 instances

5.1.2 Clausification of Tree-like Parts

There are many real-world instances that are not tree-like. However, a significant subset of the xor-constraints may form a tree-like xor-constraint conjunction. For example, consider the xor-constraint conjunction ϕ_{xor} whose constraint graph is shown in Figure 5.5.

Tree-like parts can be exploited by applying stronger parity reasoning techniques only on the non-tree-like parts. The tree-like parts can be handled by plain unit propagation without risk of not detecting some xor-implied literals. We found that this technique may lead to faster solving time on some instances.

In the Hitag2 benchmarks, roughly one fourth of the xor-constraints are in the tree-like part. We ran cryptominisat 2.9.2 on these instances with three modes: (i) without Gaussian elimination, (ii) with Gaussian elimination, and (iii) with Gaussian elimination and tree-like parts translated to CNF. The results are shown in Figure 5.6. Relatively costly Gaussian elimination seems to be useful only for the harder instances. There are also instance families that do not contain any tree-like parts (e.g. our known-plaintext attack on the Trivium cipher) or have very small tree-like parts (e.g. known-plaintext attack on the Grain cipher). We also conducted a similar study with Minisat 2.0 core enhanced with Subst xor-deduction system but found no runtime improvements. We can thus conclude that the potentially beneficial effect of translating tree-like parts to CNF depends both from the solver and the benchmark.

We also ran cryptominisat 2.9.2 using the same three modes on the 413 aforementioned non-tree-like SAT Competition instances. We measured the proportions of tree-like parts in these instances. The results are shown in Figure 5.7. The biggest instance with a significant tree-like part in this set has 312707 xor-constraints and 229067 of them are tree-like. Many of these instances have a large tree-like part. Figure 5.8 shows the comparison of solving time with three different modes. Using Gaussian elimination on the non-tree-like part can be very useful. However, for larger matrices, the computational overhead of Gaussian elimination is significant. By translating tree-like parts to CNF, some of these matrices can be made smaller and this results in a fairly consistent speedup.

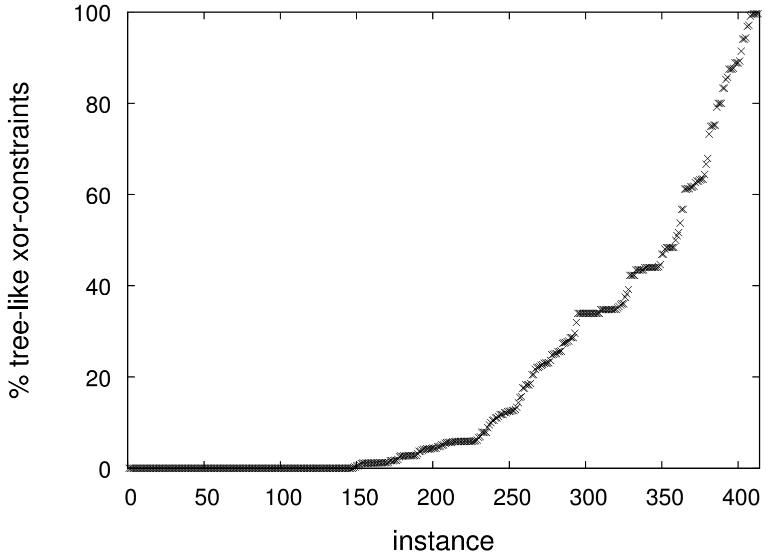


Figure 5.7. Relative tree-like part sizes of non-tree-like SAT Competition instances

5.2 When Equivalence Reasoning Deduces All Xor-Implied Literals

As shown earlier, unit propagation cannot detect all xor-implied literals on many real-world instances. Now we study how equivalence reasoning can be characterized as a structural property of the constraint graph. By exploiting this connection, we develop a fast approximating test to detect whether equivalence reasoning is enough to detect all xor-implied literals.

We first establish a close connection between equivalence reasoning and cycles in the constraint graph by using the EC xor-deduction system. To simplify the following translations and related proofs, we consider a restricted class of xor-constraints.

Definition 5. *A conjunction of xor-constraints* ϕ_{xor} *is in 3-xor normal form if (i) every xor-constraint in is has exactly three variables, and (ii) each pair of xor-constraints shares at most one variable.*

Given a ϕ_{xor}, an equisatisfiable 3-xor normal form formula can be obtained by (i) eliminating unary and binary xor-constraints by unit propagation and substitution, (ii) cutting longer xor-constraints as described above, and (iii) applying the following rewrite rule:

$$(x_1 \oplus x_2 \oplus x_3 \equiv p_1) \wedge (x_2 \oplus x_3 \oplus x_4 \equiv p_2) \rightsquigarrow (x_1 \oplus x_2 \oplus x_3 \equiv p_1) \wedge (x_1 \oplus x_4 \equiv p_1 \oplus p_2)$$

In 3-xor normal form, $\oplus$-Conflict is actually a shorthand for two applica-

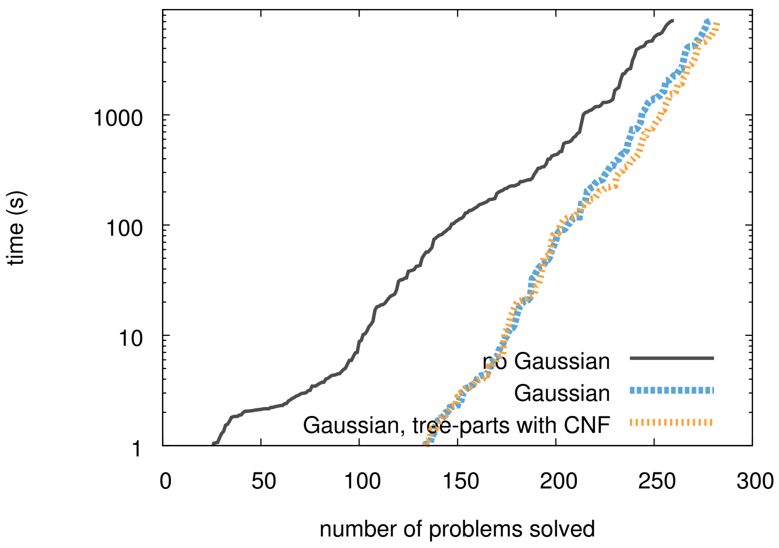


Figure 5.8. Run-times of non-tree-like SAT Competition instances

tions of $\oplus$-Imply and one application of Conflict, so the rule $\oplus$-Imply succinctly characterizes equivalence reasoning. We now prove that the rule $\oplus$-Imply is closely related to the cycles in the constraint graphs.

Definition 6. *An xor-cycle is a conjunction of xor-constraints having the form* $(x_1 \oplus x_2 \oplus y_1 \equiv p_1) \wedge \cdots \wedge (x_{n-1} \oplus x_n \oplus y_{n-1} \equiv p_{n-1}) \wedge (x_1 \oplus x_n \oplus y_n \equiv p_n)$*, abbreviated with* $XC(\langle x_1, ..., x_n \rangle, \langle y_1, ..., y_n \rangle, p)$ *where* $p = p_1 \oplus ... \oplus p_n$*. We call* $x_1, ..., x_n$ *the inner variables and* $y_1, ..., y_n$ *the outer variables of the xor-cycle.*

Example 9. *The CNF-xor instance shown in Figure 5.5 has one xor-cycle* $(a \oplus b \oplus c \equiv \bot) \wedge (c \oplus d \oplus e \equiv \top) \wedge (b \oplus d \oplus j \equiv \top)$*, where* b, c, d *are the inner and* a, e, j *the outer variables.*

A key observation is that the $\oplus$-Imply rule can be used to deduce an xor-implied literal *exactly when there is an xor-cycle* such that the values of the outer variables except for one are already deduced:

Lemma 8 (Lem. 1 of Laitinen *et al.* [2014a]). *Assume an* EC*-derivation* G *on* $\psi = \phi_{\text{xor}} \wedge l_1 \wedge ... \wedge l_k$*, where* ϕ_{xor} *is a 3-xor normal form xor-constraint conjunction. There is an extension* G' *of* G *where an xor-constraint* $(y \equiv p \oplus p'_1 \oplus ... \oplus p'_{n-1})$ *is derived using* $\oplus$*-Imply on the xor-constraints* $\{(x_1 \oplus x_2 \equiv p_1 \oplus p'_1), ..., (x_{n-1} \oplus x_n \equiv p_{n-1} \oplus p'_{n-1}), (x_1 \oplus x_n \oplus y \equiv p_n)\}$ *if and only if there is an xor-cycle* $XC(\langle x_1, ..., x_n \rangle, \langle y_1, ... y_{n-1}, y \rangle, p) \subseteq \phi_{\text{xor}}$ *where* $p = p_1 \oplus ... \oplus p_n$ *such that* $\psi \vdash_{\text{EC}} (y_i \equiv p'_i)$ *for each* $y_i \in \{y_1, ..., y_{n-1}\}$*.*

The presence of xor-cycles in the problem implies that equivalence reasoning might be useful, but does not give any indication of whether it is enough to always deduce all xor-implied literals. Again, we do not know any easy way to detect whether a given xor-constraint conjunction is Subst-deducible (or equivalently, EC-deducible). However, we can obtain a very fast structural test for approximating EC-deducibility as shown and analyzed in the following.

We say that a 3-xor normal form xor-constraint conjunction ϕ_{xor} is *cycle-partitionable* if there is a partitioning $(V_{\text{in}}, V_{\text{out}})$ of $\text{vars}(\phi_{\text{xor}})$ such that for each xor-cycle $XC(X, Y, p)$ in ϕ_{xor} $X \subseteq V_{\text{in}}$ and $Y \subseteq V_{\text{out}}$. That is, there should be no variable that appears as an inner variable in one xor-cycle and as an outer variable in another. For example, the instance in Figure 5.5 is cycle-partitionable as $(\{b, c, d\}, \{a, e, f, ..., m\})$ is a valid cycle-partition. On the other hand, the one in Figure 5.1(b) is not cycle-partitionable (although it is UP-deducible and thus EC-deducible). If such cycle-partition can be found, then equivalence reasoning is enough to always deduce all xor-implied literals.

Theorem 9 (Thm. 2 of Laitinen *et al.* [2014a]). *If a 3-xor normal form xor-constraint conjunction ϕ_{xor} is cycle-partitionable, then it is* Subst*-deducible (and thus also* EC*-deducible).*

Detecting whether a cycle-partitioning exists can be efficiently implemented with a variant of Tarjan's algorithm for strongly connected components.

To evaluate the accuracy of the technique, we applied it to the SAT Competition instances. The results are shown in the "cycle-partitionable" and "probably Subst" columns in Figure 5.2(a), where the latter gives the number of instances for which our random testing procedure described in Section 5.1.1 was not able to show that the instance is not Subst-deducible. We see that the accuracy of the cycle-partitioning test is (probably) not perfect in practice although for some instance families it works very well.

6. Decomposing Parity Constraints

In the DPLL(XOR) setting, the problem instance to be solved is given in two parts: the CNF-part and the xor-part. The top-level SAT solver takes care of the CNF-part and the associated xor-reasoning module handles the xor-part. We now explore some techniques to improve solving performance by decomposing the xor-part into distinct xor-constraint conjunctions. Decomposition methods can be used to improve performance in various ways, but here our primary aim is to improve the memory usage of Gauss-Jordan elimination when using dense matrix representation.

In this chapter, we develop a new decomposition theorem that sometimes allows us to split the xor-constraint part into components that can be handled individually. The technique exploits a variant of "biconnected components" by splitting the xor-part into components that are connected to each other by single cut variables. We prove that if we can provide full propagation for each of the components, we have full propagation for the whole xor-part as well. This is useful for reducing memory usage and improving propagation efficiency when using Gauss-Jordan elimination on dense matrices. We show that the memory usage sometimes can be optimized further by eliminating some of the variables occurring only in the xor-part while preserving the biconnected component decomposition. We develop also a generalization of the decomposition theorem that allows to decompose the instance using a set of cut variables.

Dechter [2003] describes a CSP solution technique in which a CSP is decomposed into an acyclic graph of sub-problems which can then be solved in polynomial time using a message-passing algorithm. Bozzano *et al.* [2006] suggest an efficient method to solve Satisfiability Modulo Theories problems involving two theories T_1 and T_2 by using a SAT solver as a top-level search engine to enumerate truth assignments on literals in $T_1 \cup T_2$ and checking consistency of each theory in isolation. As such, this does not guarantee that the two partial models are mutually consistent, but the solver guarantees mutual consistency on demand by finding a truth assignment to all the possible equalities between the “interface” variables, that is, variables belonging both to T_1 and T_2. While in our work we only consider a single theory involving xor-constraints, the decomposition theorem presented in Section 6.3 and applied in Section 7.5.3 is similar to the methods presented in Bozzano *et al.* [2006] in the sense that the conjunction of xor-constraints is decomposed into two parts and can be solved in isolation provided that mutual consistency is achieved by communicating all the possible linear combination involving shared variables.

6.1 Biconnected Component Decomposition

If dense representation for matrices is used, the worst-case memory use for Gauss or Gauss-Jordan elimination on $\mathrm{O}(ne)$, where n is the number of variables and e is the number of linearly independent xor-constraints in a given ϕ_{xor}. In this case, decomposing variable-disjoint sets of xor-constraints into distinct matrices can lower the memory usage.

We now present an improved decomposition technique based on biconnected components of constraint graph. The technique is formally captured in the theorem that states that full propagation can be achieved by (i) propagating values only through “cut variables”, and (ii) providing full propagation for “biconnected components” connected by cut variables. Each component can be handled by a separate xor-reasoning module and there is no need to consider equivalences or other relationships between variables in different modules.

Formally, given an xor-constraint conjunction ϕ_{xor}, we define that a *cut variable* is a variable $x \in \mathrm{vars}(\phi_{\mathrm{xor}})$ for which there is a partition $(V_{\mathrm{a}}, V_{\mathrm{b}})$ of xor-constraints in ϕ_{xor} with $\mathrm{vars}(V_{\mathrm{a}}) \cap \mathrm{vars}(V_{\mathrm{b}}) = \{x\}$; such a partition $(V_{\mathrm{a}}, V_{\mathrm{b}})$ is called an *x-cut partition* of ϕ_{xor}. The *biconnected components* of ϕ_{xor} are defined to be the equivalence classes in the reflexive and transitive closure of the relation $\{(D, E) \mid D \text{ and } E \text{ share a non-cut variable}\}$ over the xor-constraints in ϕ_{xor}.

For graphs, (i) a *connected component* of constraint graph G is a maximal connected subgraph of G, and (ii) a *cut vertex* is a vertex of G is a vertex in it whose removal will break a connected component of G into two or more connected components, and (iii) a *biconnected component* of G is a maximal biconnected subgraph (a graph is biconnected if it is connected

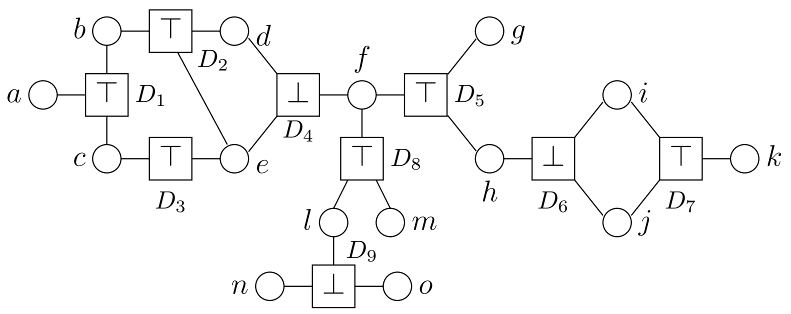


Figure 6.1. The constraint graph of the conjunction ϕ_{xor} in Example 10

and removing any vertex leaves the graph connected).

Example 10. *The constraint graph of the conjunction* $\phi_{\text{xor}} = (a \oplus b \oplus c \equiv \top) \wedge (b \oplus d \oplus e \equiv \top) \wedge (c \oplus e \equiv \top) \wedge (d \oplus e \oplus f \equiv \bot) \wedge (f \oplus g \oplus h \equiv \top) \wedge (h \oplus i \oplus j \equiv \bot) \wedge (i \oplus j \oplus k \equiv \top) \wedge (f \oplus l \oplus m \equiv \top) \wedge (l \oplus n \oplus o \equiv \bot)$ *is shown in Figure 6.1. The cut vertices of it are* D_1*,* D_4*,* f*,* D_5*,* h*,* D_6*,* D_7*,* D_8*,* l*, and* D_9*. Its biconnected components are the subgraphs induced by the vertex sets* $\{a, D_1\}$*,* $\{D_1, b, D_2, d, c, D_3, e, D_4\}$*,* $\{D_4, f\}$*, and so on. Observe that the biconnected components are not vertex-disjoint.*

Since constraint graphs have vertices also for xor-constraints, the biconnected components of a constraint graph G for ϕ_{xor} do not directly correspond to the biconnected components of ϕ_{xor}. However, the cut vertices of G correspond exactly to the cut variables of ϕ_{xor}. Therefore, we have linear-time algorithm for computing the biconnected components of ϕ_{xor}:

1. Build the constraint graph G for ϕ_{xor}.
2. Use an algorithm by Hopcroft and Tarjan [1973] to compute the biconnected components of G in linear time; as a byproduct the algorithm gives cut vertices as well.
3. Build the biconnected components of ϕ_{xor} by putting two xor-constraints in the same component if they share a non-cut variable.

Since biconnected components are connected to each other only through cut variables, they can actually be handled by different xor-reasoning modules in the DPLL(XOR) framework. The CNF-part solver can communicate values of cut variables between xor-reasoning modules; if one xor-reasoning module implies the value of a cut variable, it can be communicated to another xor-reasoning module as an xor-assumption. The next theorem states that this kind of decomposition preserves full propagation provided that each xor-reasoning module can provide full propagation for its component.

Theorem 10 (Thm. 4 of Laitinen *et al.* [2012]). *Let* $(V_{\mathrm{a}}, V_{\mathrm{b}})$ *be an* x*-cut partition of* ϕ_{xor}. *Let* $\phi^{\mathrm{a}}_{\mathrm{xor}} = \bigwedge_{D \in V_{\mathrm{a}}} D$, $\phi^{\mathrm{b}}_{\mathrm{xor}} = \bigwedge_{D \in V_{\mathrm{b}}} D$, *and* $l_1, ..., l_k, \hat{l} \in \mathrm{lits}(\phi_{\mathrm{xor}})$. *Then the following facts hold:*

- *If* $\phi_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k$ *is unsatisfiable, then*

  1. $\phi^{\mathrm{a}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k$ *or* $\phi^{\mathrm{b}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k$ *is unsatisfiable; or*
  2. $\phi^{\mathrm{a}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \models (x \equiv p_x)$ *and* $\phi^{\mathrm{b}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \models (x \equiv p_x \oplus \top)$ *for some* $p_x \in \{\bot, \top\}$.

- *If* $\phi_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k$ *is satisfiable and* $\phi_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \models \hat{l}$, *then*

  1. $\phi^{\mathrm{a}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \models \hat{l}$ *or* $\phi^{\mathrm{b}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \models \hat{l}$*; or*
  2. $\phi^{\mathrm{a}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \models (x \equiv p_x)$ *and* $\phi^{\mathrm{b}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \wedge (x \equiv p_x) \models \hat{l}$*; or*
  3. $\phi^{\mathrm{b}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \models (x \equiv p_x)$ *and* $\phi^{\mathrm{a}}_{\mathrm{xor}} \wedge l_1 \wedge ... \wedge l_k \wedge (x \equiv p_x) \models \hat{l}$.

We observe the following: some biconnected components can be singleton sets. Basic unit propagation provides full propagation for such sets. Such singleton components originate from “tree-like” parts of ϕ_{xor}: the trees can be “outermost” (constraints D_8 and D_9 in Figure 6.1) or between two non-tree-like components (D_5 in Figure 6.1). Thus our new result in a sense subsumes the result in Section 5.1.2 (originally in [III]) where we suggest clausification of “outermost” tree-like parts.

6.1.1 Experimental Evaluation

To evaluate the relevance of detecting biconnected components, we studied the SAT Competition benchmark instances. We first examine how the memory usage can be improved by removing (i) tree-like xor-constraints and (ii) storing each biconnected component in a separate matrix. Figures 6.2 and 6.3 show the reduction in memory usage when using dense matrix representation to represent the xor-constraints. As already shown

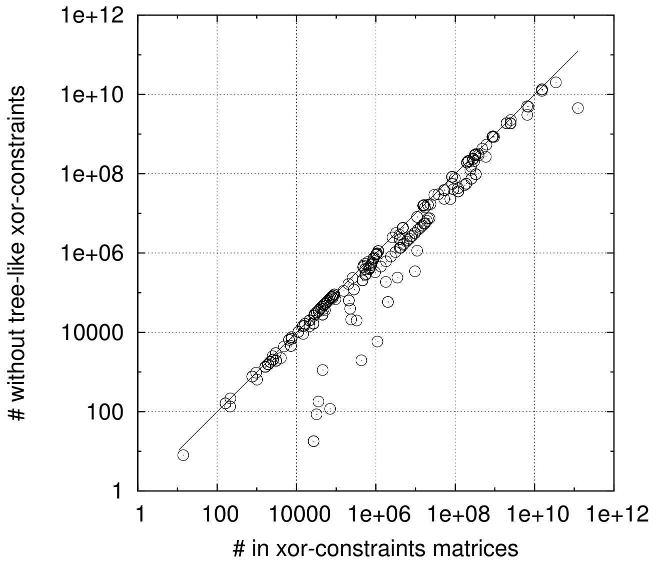


Figure 6.2. Reduction in memory usage for dense matrix representation when tree-like xor-constraints are removed

in the previous chapter, a significant proportion of xor-constraints in these competition instances are tree-like and performing additional reasoning beyond unit propagation cannot be used to detect more implied literals. Removing these tree-like xor-constraints from Gauss-Jordan matrices reduces the memory usage greatly. An additional reduction in memory usage is obtained by storing each biconnected component in a separate matrix.

We ran minisat 2.0 core augmented with four different xor-reasoning modules (unit propagation (UP), equivalence reasoning (Subst), incremental Gauss-Jordan elimination (IGJ), and a variant of IGJ exploiting biconnected components) and cryptominisat 2.9.2 on these instances at most hour on 12-core Intel X5650 with 48GB memory (2 GB memory limit per task)[1].

Figure 6.4 shows the number of instances solved with respect to the number of heuristic decisions. Unit propagation and equivalence reasoning perform similarly on these instances[2]. Incremental Gauss-Jordan solves a substantial number of the instances almost instantly and also manages to solve more instances in total. The solver cryptominisat 2.9.2

[1]The solver configuration IGJ performs better in Section 3.5 due to higher memory limit and newer CPU.

[2]In these experiments, the solver configuration with Subst translates all xor-constraints to CNF to resort to equivalence reasoning only when unit propagation is saturated, so the results differ from the ones presented in Section 3.5

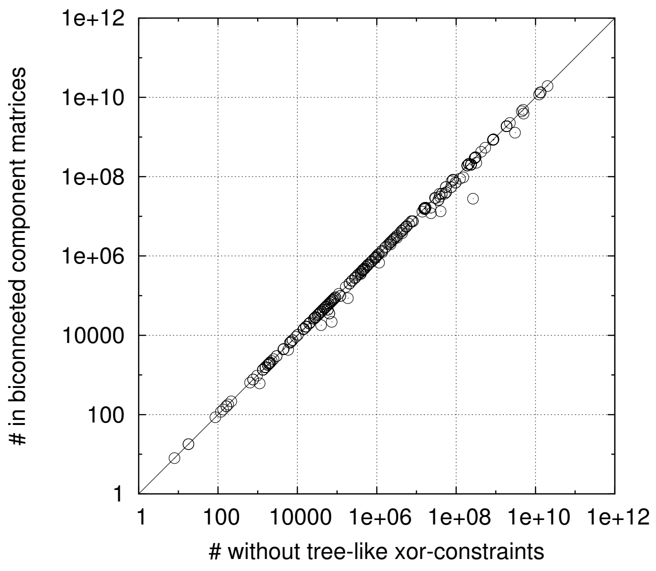


Figure 6.3. Reduction in memory usage for dense matrix representation when only biconnected components are counted in SAT 2005-2011 competition instances. Although the difference seems negligible in logarithmic scale, the memory consumption is reduced by additional 13.5% on average in 110 instances having multiple biconnected components.

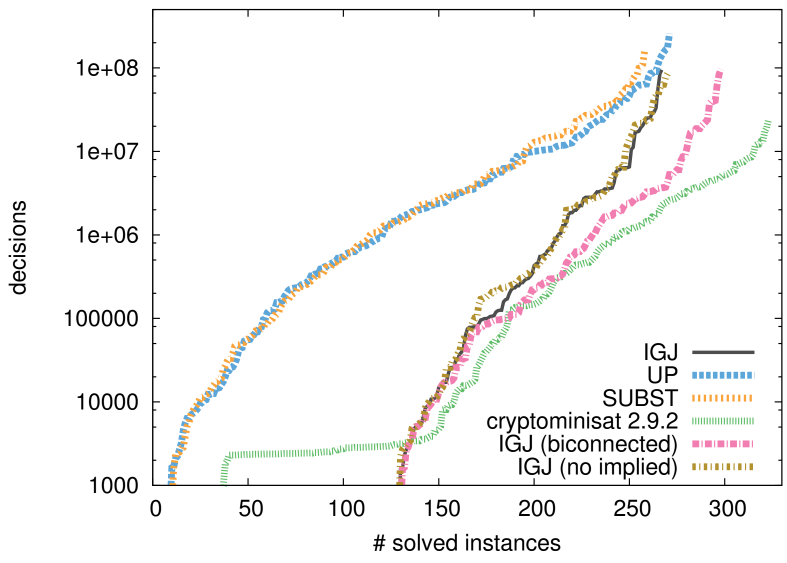


Figure 6.4. Number of SAT 2005-2011 competition instances solved with respect to decisions

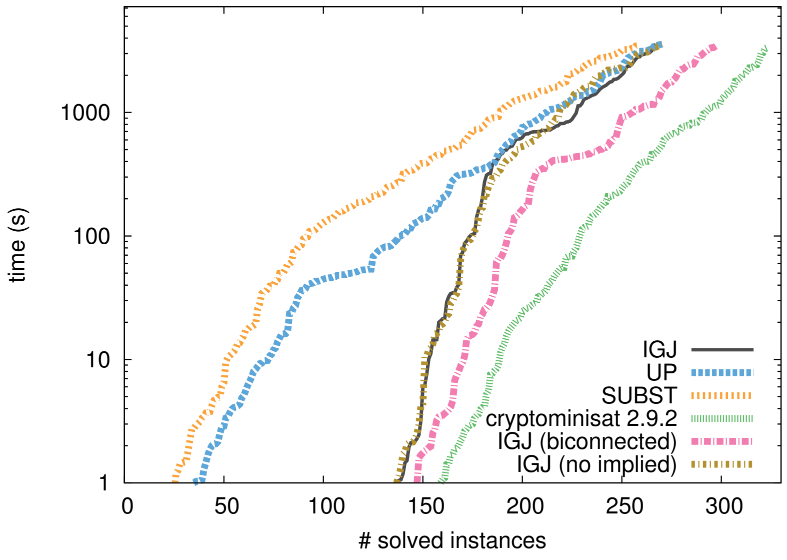


Figure 6.5. Number of SAT 2005-2011 competition instances solved with respect to time

performs very well on these instances. Figure 6.5 shows the number of instances solved with respect to time. Since equivalence reasoning does not reduce the number of decisions, the computational overhead is reflected in the slowest solving time. Incremental Gauss-Jordan is computationally more intensive but complete parity reasoning pays off on these instances leading to fastest solving compared to our other xor-reasoning modules. Omitting tree-like xor-constraints from Gauss-Jordan matrices and splitting biconnected components into separate matrices lowers the overall memory requirement enough to solve more instances within the 2 GB memory limit. In [IV], we reported a reduction in the solving time by splitting biconnected components into separate matrices. However, in our current implementation the overhead per Gauss-Jordan matrix is significant and the solving time was not greatly improved. To illustrate the effect of implied literals deduced by Gauss-Jordan, we also ran a solver using Gauss-Jordan only to detect conflicts and otherwise resorting to unit propagation. More instances are solved when all implied literals are deduced.

Biconnected components may be exploited even without modifying the solver. The solver cryptominisat accepts as its input a mixture of clauses and xor-constraints. When Gaussian elimination is used, the solver stores each connected component in a separate matrix. By translating each singleton biconnected component into CNF, some non-trivial biconnected

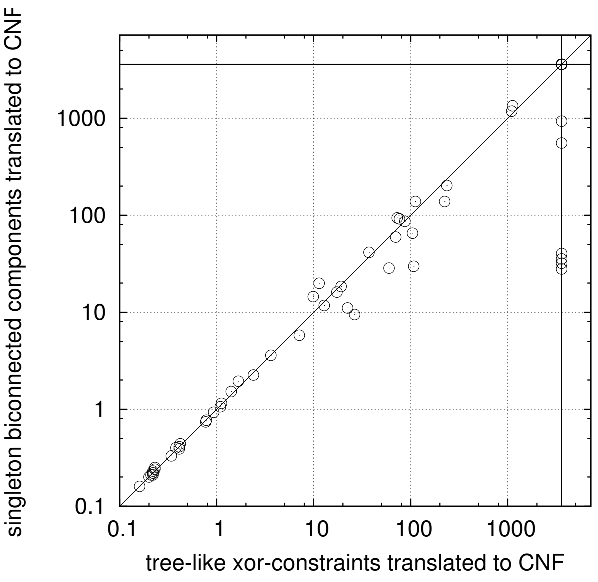


Figure 6.6. Effect in solving time for cryptominisat when singleton biconnected components in SAT competition instances are translated to CNF

components may become connected components and are then placed into separate matrices improving memory usage. We considered the 110 SAT competition instances with multiple biconnected components and found 60 instances where some biconnected components could be separated by translating singleton biconnected components to CNF. Figures 6.6 and 6.7 show the effect of the translation in the number of decisions and solving time. The solver cryptominisat 2.9.2 solves 44 of the unmodified instances. After the translation, cryptominisat 2.9.2 is able to solve 50 instances and slightly faster.

6.2 Eliminating XOR-Internal Variables

A CNF-xor formula $\phi_{\text{or}} \wedge \phi_{\text{xor}}$ may have *xor-internal* variables occurring only in ϕ_{xor}. Such variables can be eliminated from ϕ_{xor} as described in Laitinen *et al.* [2010] by substituting them with their "definitions"; e.g. if $x_1 \oplus x_2 \oplus x_3 \equiv \top$ is an xor-constraint where x_1 is an xor-internal variable, then remove the xor-constraint and replace every occurrence of x_1 in all the other xor-constraints by $x_2 \oplus x_3 \oplus \top$. When using dense matrix representation, the matrices can be made more compact by eliminating xor-internal variables. For instance, one of our Trivium benchmark instances has 5900 xor-internal variables out of 11484 variables and 8590

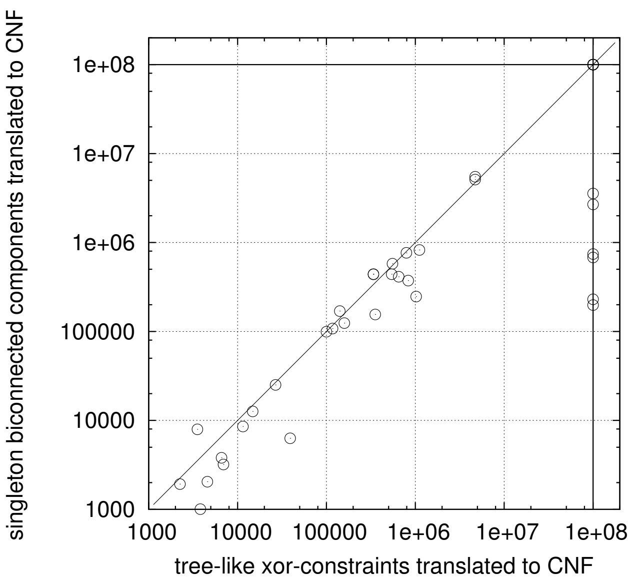


Figure 6.7. Effect in solving time for cryptominisat when singleton biconnected components in SAT competition instances are translated to CNF

parity constraints in two connected components. The total number of elements in the matrices is 55×10^6 elements. By eliminating all xor-internal variables this can be reduced to 8×10^6 elements. The instance has three biconnected components (as all of our Trivium instances) and storing them in separate matrices requires 33×10^6 elements in total. But, if a cut variable connecting the biconnected components is xor-internal, it is eliminated and the two biconnected components are merged into one bigger biconnected component. To preserve biconnected components, only the variables occurring in a single biconnected component and not in the CNF-part should be eliminated. There are 5906 such variables in the instances and after the elimination the total number of elements in three matrices is 5×10^6. Figures 6.8 and 6.9 show the effect of eliminating such variables in our Trivium instances. Unit propagation benefits from elimination of xor-internal variables. Fewer watched literals (variables) are needed for longer xor-constraints to detect when an implied literal can be deduced. The solver configuration using incremental Gauss-Jordan elimination manages to solve all of our benchmark instances with reduced solving time.

Related work. The solver cryptominisat by Soos *et al.* [2009]; Soos [2010] eliminates some xor-internal variables that occur only at most two xor-constraints. Variables having only one occurrence are dependent and are

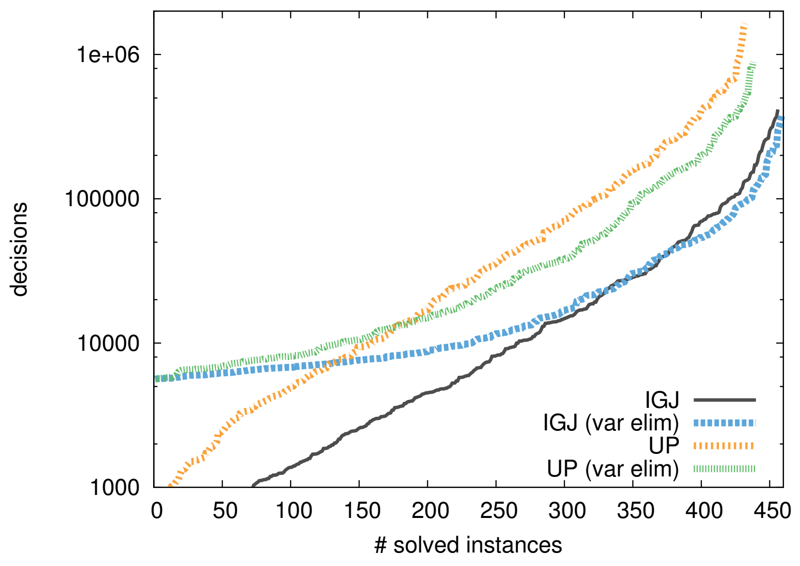


Figure 6.8. Effect of eliminating xor-internal variables while preserving biconnected components on the number of decisions on smaller Trivium benchmark set (459 instances)

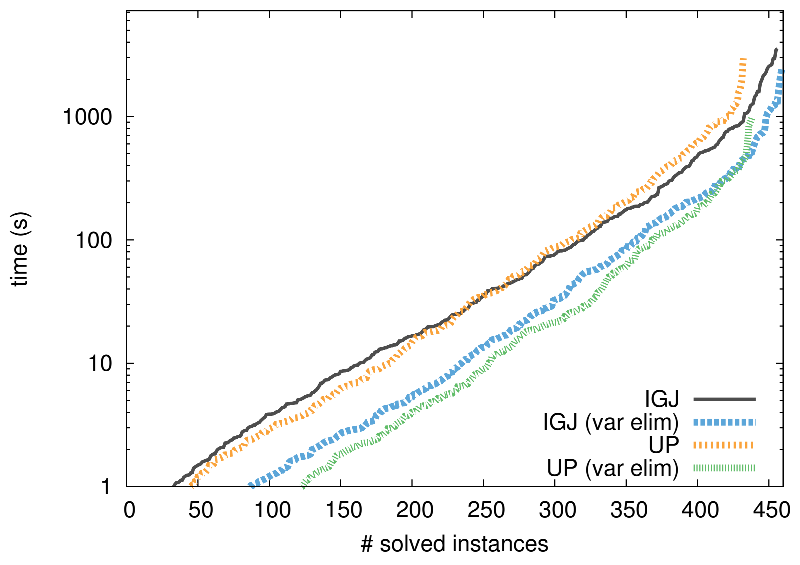


Figure 6.9. Effect of eliminating xor-internal variables while preserving biconnected components on the solving time on smaller Trivium benchmark set (459 instances)

removed along with the corresponding xor-constraint from the instance. Variables with two occurrences are eliminated by adding linear combination of the two corresponding xor-constraint in the instance.

6.3 N-cut decomposition

The new decomposition technique is a generalization of the method presented in the previous section, which states that, in order to guarantee full propagation, it is enough to (i) propagate only values through "cut variables", and (ii) have full propagation for the "biconnected components". Now we extend the technique to larger cuts. Given an xor-constraint conjunction ϕ_{xor}, a *cut variable set* is a set of variables $X \subseteq \text{vars}(\phi_{\text{xor}})$ for which there is a partition $(V_{\text{a}}, V_{\text{b}})$ of xor-constraints in ϕ_{xor} with $\text{vars}(V_{\text{a}}) \cap \text{vars}(V_{\text{b}}) = X$; such a partition $(V_{\text{a}}, V_{\text{b}})$ is called an *X-cut partition* of ϕ_{xor}. If full propagation can be guaranteed for both sides of an X-cut partition, then communicating the implied linear combinations involving cut variables is enough to guarantee full propagation for the whole instance:

Theorem 11 (Thm. 6 of Laitinen *et al.* [2013]). *Let $(V_{\text{a}}, V_{\text{b}})$ be an X-cut partition of ϕ_{xor}. Let $\phi^{\text{a}}_{\text{xor}} = \bigwedge_{D \in V_{\text{a}}} D$, $\phi^{\text{b}}_{\text{xor}} = \bigwedge_{D \in V_{\text{b}}} D$, and $l_1, \ldots, l_k, \hat{l} \in \text{lits}(\phi_{\text{xor}})$. Then the following facts hold:*

- *If $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$ is unsatisfiable, then*

  1. *$\phi^{\text{a}}_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$ or $\phi^{\text{b}}_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$ is unsatisfiable; or*
  2. *$\phi^{\text{a}}_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k \models (X' \equiv p)$ and $\phi^{\text{b}}_{\text{xor}} \wedge l_1 \wedge \ldots l_k \models (X' \equiv p \oplus \top)$ for some $X' \subseteq X$ and $p \in \{\top, \bot\}$.*

- *If $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$ is satisfiable and $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k \models \hat{l}$, then*

  1. *$\phi^{\text{a}}_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k \models \hat{l}$ or $\phi^{\text{b}}_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k \models \hat{l}$; or*
  2. *$\phi^{\alpha}_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k \models (X' \equiv p)$ and $\phi^{\beta}_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k \wedge (X' \equiv p) \models \hat{l}$ for some $X' \subseteq X$, $p \in \{\top, \bot\}$, $\alpha \in \{\text{a}, \text{b}\}$, and $\beta \in \{\text{a}, \text{b}\} \setminus \{\alpha\}$.*

By applying Theorem 11 we can decompose triconnected (4-connected,

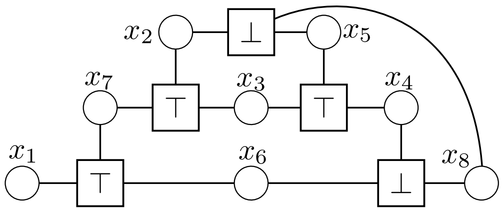


Figure 6.10. Constraint graph of an xor-constraint conjunction

5-connected, etc.) components. As with biconnected component decomposition presented in Section 6.1, this may allow lower memory usage for Gauss or Gauss-Jordan elimination on dense matrices. Also, each component may be handled by a separate xor-reasoning module. However, as components may be connected with more than one variable, implied linear combinations involving shared variables must be communicated between the xor-reasoning modules. Communicating such linear combinations may be done by introducing auxiliary variables as illustrated in the following example.

Example 11. *Consider the constraint graph in Figure 6.10. The cut variable set* $\{x_2, x_3, x_6\}$ *partitions the xor-constraints into two conjunctions* $\phi^{\mathrm{a}}_{\mathrm{xor}} = (x_1 \oplus x_6 \oplus x_7 \equiv \top) \wedge (x_2 \oplus x_3 \oplus x_7 \equiv \top)$ *and* $\phi^{\mathrm{b}}_{\mathrm{xor}} = (x_2 \oplus x_5 \oplus x_8 \equiv \perp) \wedge (x_3 \oplus x_4 \oplus x_5 \equiv \top) \wedge (x_4 \oplus x_6 \oplus x_8 \equiv \perp)$*. Note that* $\phi^{\mathrm{b}}_{\mathrm{xor}} \models (x_2 \oplus x_3 \oplus x_6 \equiv \top)$ *and* $\phi^{\mathrm{a}}_{\mathrm{xor}} \wedge (x_2 \oplus x_3 \oplus x_6 \equiv \top) \models (x_1 \equiv \top)$*. The possible implied linear combinations involving variables in the cut variable set* $\{x_2, x_3, x_6\}$ *are* $(x_2 \equiv p)$, $(x_3 \equiv p)$, $(x_6 \equiv p)$, $(x_2 \oplus x_3 \equiv p)$, $(x_2 \oplus x_6 \equiv p)$, $(x_3 \oplus x_6 \equiv p)$, $(x_2 \oplus x_3 \oplus x_6 \equiv p)$ *where* $p \in \mathbb{B}$*. By adding on both sides* $\phi^{\mathrm{a}}_{\mathrm{xor}}$ *and* $\phi^{\mathrm{b}}_{\mathrm{xor}}$ *xor-constraints* $\phi = (a_{2,3} \oplus x_2 \oplus x_3 \equiv \perp) \wedge (a_{2,6} \oplus x_2 \oplus x_6 \equiv \perp) \wedge (a_{3,6} \oplus x_3 \oplus x_6 \equiv \perp) \wedge (a_{2,3,6} \oplus x_2 \oplus x_3 \oplus x_6 \equiv \perp)$ *where* $a_{2,3}$, $a_{2,6}$, $a_{3,6}$, $a_{2,3,6}$ *are new auxiliary variables, all possible implied linear combinations can be communicated by assigning the variables* $x_2, x_3, x_6, a_{2,3}, a_{3,6}, a_{2,6}, a_{2,3,6}$ *appropriately, e.g.* $\phi^{\mathrm{b}}_{\mathrm{xor}} \wedge \phi \models (a_{2,3,6} \equiv \top)$ *and* $\phi^{\mathrm{a}}_{\mathrm{xor}} \wedge \phi \wedge (a_{2,3,6} \equiv \top) \models (x_1 \equiv \top)$.

The number of auxiliary variables is $2^n - n - 1$ where n is the size of the cut variable set, so only relatively small cut variable sets can be used in practice. Theorem 11 may be iteratively applied multiple times to partition the formula further provided that auxiliary variables are considered as original variables in subsequent decompositions.

In the following, we apply the decomposition method using a tree decomposition to obtain small (candidate) cut variable sets. Formally, a *tree decomposition* of a graph $G = \langle V, E \rangle$ is a pair $\langle X, T \rangle$, where $X = \{X_1, \ldots, X_n\}$ is a family of subsets of V, and T is a tree whose nodes are the subsets X_i,

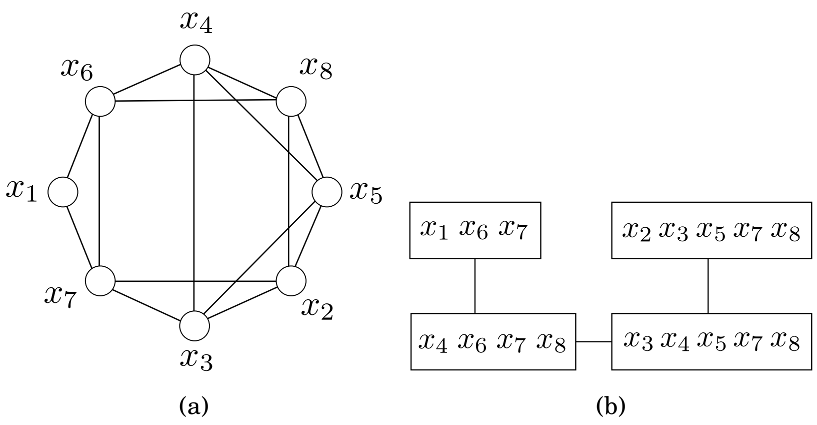


Figure 6.11. (a) primal graph, (b) tree decomposition for the primal graph

satisfying the following properties: (i) $V = X_1 \cup \cdots \cup X_n$, (ii) if $\langle v, v' \rangle \in E$, then $\{v, v'\} \subseteq X_i$ for at least one $X_i \in X$. and (iii) if a node v is in two sets X_i and X_j, then all nodes in the path between X_i and X_j contain v. The width of a tree decomposition is the size of its largest set X_i minus one. The *treewidth* $\mathrm{tw}(G)$ of a graph G is the minimum width among all possible tree decompositions of G.

Each pair of adjacent nodes in a tree decomposition defines a cut variable set, so a tree decomposition can be used to obtain a list of small cut variable sets. The *primal graph* for an xor-constraint conjunction ϕ_{xor} is a graph such that the nodes correspond to the variables of ϕ_{xor} and there is an edge between two variable nodes if and only if both variables have an occurrence in the same xor-constraint.

Example 12. *Figure 6.11(a) shows the primal graph for the xor-constraint conjunction illustrated in Figure 6.10 and Figure 6.10 shows a tree decomposition for the primal graph.*

6.3.1 Experimental Evaluation

We first studied how the memory usage of the IGJ xor-deduction system can be improved by applying N-cut decomposition in a controlled test environment when using dense matrix representation to manipulate xor-constraints. We created an artificial CNF-xor instance that consists of 201 parts that form a chain where consecutive parts share 63 variables out of which six are "interface" variables and $57 = \binom{6}{2} + \cdots + \binom{6}{6}$ are auxiliary variable tracking the relevant linear combinations of the interface variables. The parts consist of xor-constraints with three or four vari-

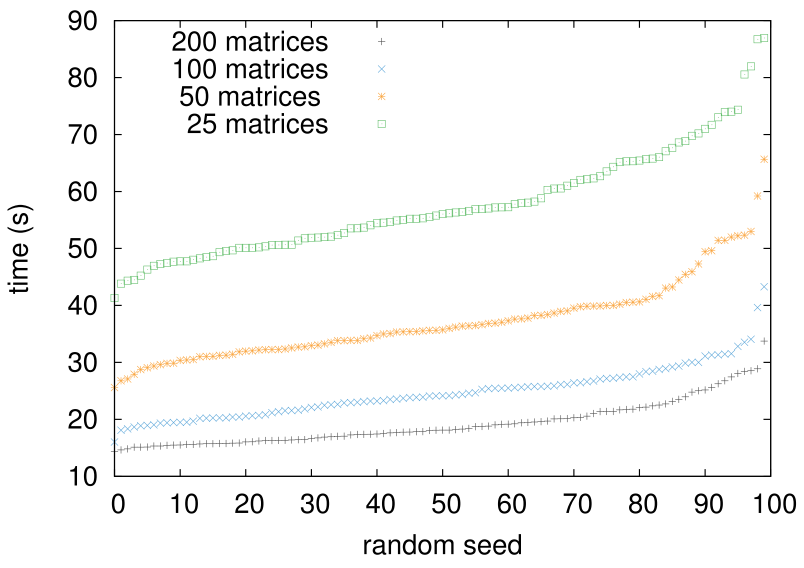


Figure 6.12. Time spent solving an artificial CNF-xor instance decomposed into independent dense matrices using N-cut decomposition. The instance is solved with 100 different random seeds.

# matrices	25	50	100	200
Median solving time (s)	56.0	35.7	24.1	18.1
Combined matrix sizes ($\times 10^6$)	21814	10930	5488	2767
Speedup w.r.t 25 matrices	1.0	1.57	2.32	3.09

Figure 6.13. Results on solving an artificial CNF-xor instance decomposed using N-cut decomposition.

ables. One part, whose xor-constraints are straightforwardly translated to CNF, has $15 \times 10 + 2 \times 63 = 276$ variables and xor-constraints. The rest, 200 parts, have $60 \times 60 + 2 \times 63 = 3726$ variables and xor-constraints. The instance is unsatisfiable. The structure of the instance makes it possible to partition it into separate matrices. Figure 6.12 shows solving time with different random seeds when the instance is decomposed into 25, 50, 100, and 200 matrices. Figure 6.13 shows median solving times, combined matrix sizes and relative speedup with respect to solving the instance with 25 matrices. The memory usage is almost halved by doubling the number of matrices. The instance is crafted in such a way that most of the solving time is spent on the xor-part, so the results indicate how much the performance can improved in near-optimal conditions. Additional matrices incur some overhead, which explains why the relative speedup grows slower as the number of matrices is increased.

We studied how much dense matrix representation memory usage could be improved in SAT Competition instances using N-cut decomposition. We applied the junction tree algorithm described in Pearl [1982] to obtain a tree decomposition of the primal graph and a list of small cut variable sets as described in Section 6.3. Each cut variable set involving at most eight variables was tested and the instance was partitioned using the cut variable set giving the highest reduction in total memory usage. Appropriate auxiliary “interface” variables were added to communicate relevant linear combinations of variables shared between the resulting parts. The resulting parts were partitioned further using the unused cut variable sets until the reduction in combined matrix sizes was less than 10000. Figure 6.14 shows the reduction in memory usage compared to original instances where tree-like xor-constraints are removed and xor-clusters are stored in individual matrices. Biconnected component decomposition is included for comparison. We also ran minisat with IGJ xor-deduction system to find out whether the reduction in memory usage is reflected in solving time. Unfortunately, there was no noticeable difference, which may be explained by the number of additional matrices which incur some overhead. It is left for future work to implement a more efficient version of IGJ xor-deduction system that scales better with hundreds or even thousands of matrices sharing small number of variables.

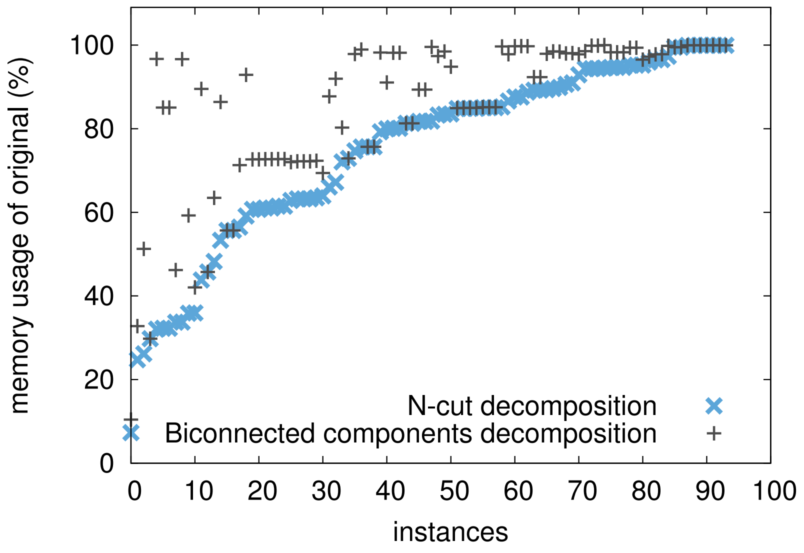


Figure 6.14. Reduction in memory usage for dense matrix representation when (i) biconnected components in SAT 2005-2011 competition instances are stored in separate matrices and (ii) N-cut decomposition is greedily applied to biconnected components. Tree-like xor-constraints are removed in the original instances.

7. Simulating Parity Reasoning

Stronger parity reasoning may prune the search space effectively but at the expense of high computational overhead, so resorting to simpler but more efficiently implementable systems, e.g. unit propagation, may lead to better performance. In this chapter, we study to what extent such simpler systems can simulate stronger parity reasoning engines in the DPLL(XOR) framework. Instead of developing yet another propagation engine and assessing it through an experimental comparison, we believe that useful insights can be acquired by considering unanswered questions on how some existing propagation engines and proof systems relate to each other on a more fundamental level. Several experimental studies have already shown that SAT solvers extended with parity reasoning engines can outperform unmodified solvers on some instances families, so we focus on more general results on the relationships between resolution, unit propagation, equivalence reasoning, parity explanations, and Gauss-Jordan elimination.

We show that resolution can simulate equivalence reasoning efficiently, which raises a question whether significant reductions in solving time can be gained by integrating specialized equivalence reasoning in a SAT solver since in theory it does not strengthen the underlying proof system of the SAT solver. In practice, though, the performance of the SAT solver is largely governed by variable selection and other heuristics that are likely to be non-optimal, which may justify the pragmatic use of equivalence reasoning.

Although equivalence reasoning alone is not enough to cross the "exponential gap" between resolution and Gauss-Jordan elimination, another light-weight parity reasoning technique comes intriguingly close at simulating complete parity reasoning. We show that parity explanations on nondeterministic unit propagation derivations can simulate Gauss-

Jordan elimination on a restricted yet practically relevant class of xor-constraint conjunctions. Choosing assumptions and unit propagation steps nondeterministically may not be possible in an actual implementation with greedy propagation strategies. However, as observed in Section 4.4, the simulation may still work in an actual implementation to some degree provided that parity explanations are stored as learned xor-constraints.

Additional xor-constraints can also be added to the formula in a preprocessing step in order to enable unit propagation to deduce more implied literals, which has the benefit of not requiring modifications to the SAT solver. The connection between equivalence reasoning and xor-cycles enables us to consider a potentially more efficient way to implement equivalence reasoning. By enumerating these cycles and adding a new linear combination of the original constraints for each such cycle to the instance, we can achieve an instance in which unit propagation simulates equivalence reasoning. As there may be an exponential number of such cycles, we develop another translation to simulate equivalence reasoning with unit propagation. The translation is polynomial as new variables are introduced; we prove that if new variables are not allowed, then there are instances families for which polynomially sized simulation translations do not exists. translation can be optimized by adding only a selected subset of the new parity constraints. Finally, we also present a translation that enables unit propagation to simulate parity reasoning systems stronger than equivalence reasoning through the use of additional xor-constraints on auxiliary variables. The translation takes into account the structure of the original conjunction of xor-constraints and can produce compact formulas for sparsely connected instances. Using the translation to simulate full Gauss-Jordan elimination with plain unit propagation requires an exponential number of additional xor-constraints in the worst case. Recently, it has been shown in Gwynne and Kullmann [2013] that a conjunction of xor-constraints does not have a polynomial-size "arc consistent" CNF-representation, which implies it is not feasible to simulate Gauss-Jordan elimination by unit propagation in the general case. On many instances, though, better solver performance can be obtained by simulating a weaker parity reasoning system as it reduces the size of the translation substantially. By applying our previous results on detecting whether unit propagation or equivalence reasoning is enough to deduce all implied literals, the size of the translation can be optimized further. The experimental evaluation suggests that the translation can improve overall solving performance on some instances.

7.1 Resolution does not Simulate Parity Explanations

Parity explanations presented in Chapter 4 can be shorter than implicative explanations making them potentially more effective in pruning the search space of the CDCL SAT solver. Now we show that this is the case at least in theory; parity explanations can be used to refute some hard formulas that do not have polynomial-size refutation proofs.

We use the "parity graph" formulas defined by Urquhart [1987]. A *parity graph* is an undirected, connected, edge-labeled graph $G = \langle G, E\rangle$ where each node $v \in V$ is labeled with a *charge* $c(v) \in \{\bot, \top\}$ and each edge $\langle u, v\rangle \in E$ is labeled with a distinct variable. The *total charge* $c(G) = \bigoplus_{v \in V} c(v)$ of a parity graph G is the parity of all node charges. Given a node v, we define the xor-constraint $\alpha(v) = q_1 \oplus \cdots \oplus q_n \equiv c(v)$, where $q_1, \ldots, q_n$ are the variables used as labels in the edges connected to v, and $\mathrm{xorclauses}(G) = \bigwedge_{v \in V} \alpha(v)$. For an xor-constraint C over n variables, let $\mathrm{cnf}(C)$ denote the equivalent CNF formula, i.e. the conjunction of 2^{n-1} clauses with n literals in each. Define $\mathrm{clauses}(G) = \bigwedge_{v \in V} \mathrm{cnf}(\alpha(v))$.

As proven in Lemma 4.1 in Urquhart [1987], $\mathrm{xorclauses}(G)$ and $\mathrm{clauses}(G)$ are unsatisfiable if and only if $c(G) = \top$. Parity graphs can be used to derive formulas that have large resolution proofs: there is an infinite sequence $G_1, G_2, \ldots$ of degree-bounded parity graphs such that $c(G_i) = \top$ for each i and the following holds:

Lemma 12 (Thm. 5.7 of Urquhart [1987]). *There is a constant $c > 1$ such that for sufficiently large m, any resolution refutation of $\mathrm{clauses}(G_m)$ contains c^n distinct clauses, where $\mathrm{clauses}(G_m)$ is of length $\mathcal{O}(n)$, $n = m^2$.*

Our important result on parity explanation follows: if G is *any* parity graph G such that $c(G) = \top$, provided that a number of xor-assumptions can be made, an empty parity explanation can be derived from $\mathrm{xorclauses}(G)$ refuting the formula with a *single* parity explanation:

Theorem 13 (Thm. 3 of Laitinen *et al.* [2014b]). *Let $G = \langle V, E\rangle$ be a parity graph such that $c(G) = \top$. There is a* UP*-refutation for $\mathrm{xorclauses}(G) \wedge q_1 \cdots \wedge q_k$ for some xor-assumptions $q_1, \ldots, q_k$, a node v with $L(v) = \bot$ in it, and a cut $W = \langle V_{\mathrm{a}}, V_{\mathrm{b}}\rangle$ for v such that $\mathit{Expl}_{\oplus}(v, W) = \top$. Thus $\mathrm{xorclauses}(G) \models (\top \Leftrightarrow \bot)$, showing $\mathrm{xorclauses}(G)$ unsatisfiable.*

It has been shown by Pipatsrisawat and Darwiche [2011] that resolution

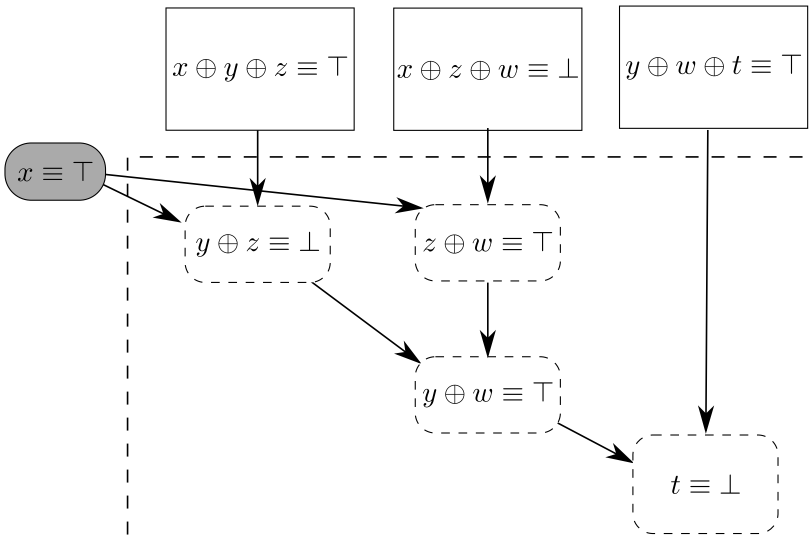


Figure 7.1. A Subst-derivation

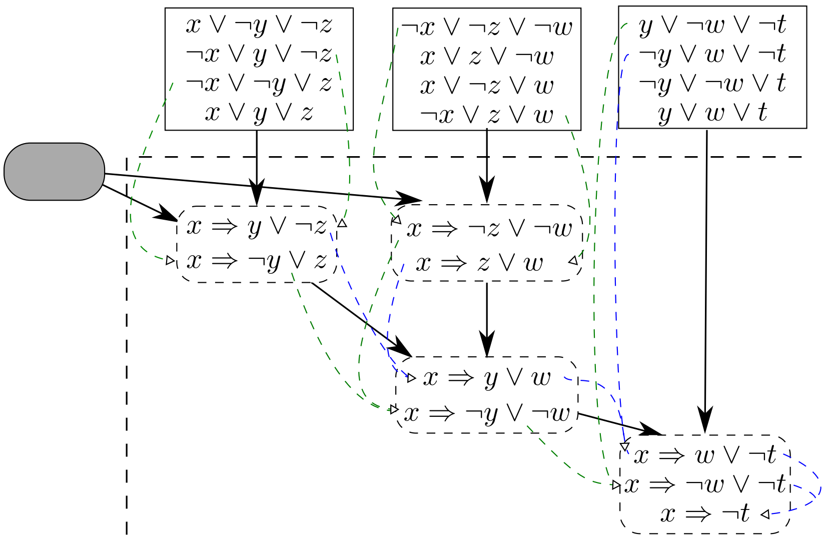
is the underlying proof system in CDCL SAT solvers. Also, provided that xor-constraints are of bounded length, unit propagation on xor-constraints can be efficiently simulated by translating the xor-constraints to CNF and propagating the CNF clauses. It follows that:

Corollary 1. *There are families of unsatisfiable CNF-xor formulas for which DPLL(XOR) using* UP*-module (i) has polynomial sized proofs if parity explanations are allowed, but (ii) does not have such if the "classic" implicative explanations are used.*

Although the CDCL SAT solver generally does not make the required xor-assumptions needed for an empty implying clause, parity explanations can be added as learned xor-constraints as explained in Section 4.2. When parity explanations are learned in this way, instances derived from parity graphs are solved fast.

7.2 Resolution Simulates Equivalence Reasoning Polynomially

As observed in the previous section, there are infinite families of xor-constraint conjunctions ϕ_{xor} such that the straightforward CNF-translation $\bigwedge_{D \in \phi_{\text{xor}}} \text{cnf}(D)$ does not have polynomial-size resolution refutation. However, such xor-constraint conjunctions can be solved in polynomial time by Gaussian elimination. Gaussian elimination can be computationally intensive for larger matrices, so computationally faster but "weaker" equiv-



$x \vee \neg y \vee \neg z$
$\neg x \vee y \vee \neg z$
$\neg x \vee \neg y \vee z$
$x \vee y \vee z$
$\neg x \vee \neg z \vee \neg w$
$x \vee z \vee \neg w$
$x \vee \neg z \vee w$
$\neg x \vee z \vee w$
$y \vee \neg w \vee \neg t$
$\neg y \vee w \vee \neg t$
$\neg y \vee \neg w \vee t$
$y \vee w \vee t$
$x \Rightarrow y \vee \neg z$
$x \Rightarrow \neg y \vee z$
$x \Rightarrow \neg z \vee \neg w$
$x \Rightarrow z \vee w$
$x \Rightarrow y \vee w$
$x \Rightarrow \neg y \vee \neg w$
$x \Rightarrow w \vee \neg t$
$x \Rightarrow \neg w \vee \neg t$
$x \Rightarrow \neg t$

Figure 7.2. A resolution derivation for an implying clause (the dotted arrows) related to Figure 7.1

alence reasoning systems have been proposed.

We now consider how relation relates to the two equally powerful xor-deduction systems Subst and EC that both implement a form of equivalence reasoning. The xor-deduction system Subst is simpler, so we use it in this study. If Subst can be used to derive an xor-constraint, then an equivalent result can be derived with resolution from the CNF-translation of the instance with a "pseudolinear" increase in the number of clauses:

Theorem 14 (Thm. 1 of Laitinen *et al.* [2013]). *Assume a* Subst*-derivation $G = \langle V, E, L \rangle$ on a conjunction ψ of xor-constraints. There is a resolution derivation π on $\bigwedge_{D \in \psi} \mathrm{cnf}(D)$ such that (i) if $v \in V$ and $L(v) \neq \top$, then the clauses $\mathrm{cnf}(L(v))$ occur in π, and (ii) π has at most $|V|2^{m-1}$ clauses, where m is the number of variables in the largest xor-constraint in ψ.*

A similar result is already observed in Li [2000b] when restricted to binary and ternary xor-constraints.

Resolution can also be used to derive clausal explanations for xor-implied literals from the CNF-translation of the instance without the use of xor-assumptions:

Theorem 15 (Thm. 2 of Laitinen *et al.* [2013]). *Assume a* Subst*-derivation $G = \langle V, E, L \rangle$ on $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k$ and a CNF-compatible cut $W = (V_{\text{a}}, V_{\text{b}})$. There is a resolution derivation π on $\bigwedge_{D \in \phi_{\text{xor}}} \mathrm{cnf}(D)$ such that (i) for each vertex $v \in V_{\text{b}}$ with $L(v) \neq \top$, π includes all the clauses in $\{\mathit{Expl}(v, W) \Rightarrow C|$*

$C \in \mathrm{cnf}(L(v))\}$, and (ii) π has at most $|V|2^{m-1}$ clauses, where m is the number of variables in the largest xor-constraint in ϕ_{xor}.

Figures 7.1 and 7.2 illustrate how to derive the implying clause $x \to \neg t$ using the construction.

Beame *et al.* [2004] formalize clause learning as a proof system and shows that when unlimited restarts are allowed, clause learning is equally powerful to resolution. Zhang and Malik [2003] describe how to check the validity of unsatisfiability claims computed by SAT solver by extracting a resolution refutation from the trace produced by the SAT solver. Resolution can thus be seen as the underlying proof system in CDCL SAT solvers. The same holds for a DPLL(XOR) with Subst or EC in the xor-reasoning module: resolution can be used to derive all implicative explanations required by the CDCL SAT solver and then considered as normal clauses when producing the resolution proof simulating the execution of the CDCL SAT solver. As shown by Pipatsrisawat and Darwiche [2011], CDCL SAT solvers can polynomially simulate resolution, so it follows that:

Corollary 2. *For CNF-xor instances with fixed width xor-constraints, the underlying proof system of a DPLL(XOR) solver using* Subst *or* EC *as the xor-reasoning module is polynomially equivalent to resolution.*

7.3 Parity Explanations (Almost) Simulate Gauss-Jordan Elimination

As observed earlier in Section 7.1, parity explanations can be used to refute some hard formulas whose CNF-translations do not have polynomial-size resolution refutations. Now we strengthen the result and show that parity explanations on UP-derivations can produce xor-constraints corresponding to the explanations produced by Gauss-Jordan elimination, provided that one can make the suitable xor-assumptions and each variable in the xor-constraint conjunction occurs at most three times.

Theorem 16 (Thm. 3 of Laitinen *et al.* [2013])**.** *Let ϕ_{xor} be a conjunction of xor-constraints such that each variable occurs in at most three xor-constraints.*

If ϕ_{xor} is unsatisfiable, then there is a UP*-derivation on $\phi_{\mathrm{xor}} \wedge y_1 \wedge \ldots \wedge y_m$ with some $y_1, \ldots, y_m \in \mathrm{vars}(\phi_{\mathrm{xor}})$, a vertex v with $L(v) = (\bot \equiv \top)$ in it, and a* *cut W for v such that $\mathit{Expl}_{\oplus}(v, W) = (\bot \equiv \bot)$ and thus $\mathit{Expl}_{\oplus}(v, W) + L(v) = (\bot \equiv \top)$.*

If ϕ_{xor} is satisfiable and $\phi_{\text{xor}} \models (x_1 \oplus ... \oplus x_k \equiv p)$, then there is a UP-*derivation on $\phi_{\text{xor}} \wedge (x_1 \equiv p_1) \wedge ... \wedge (x_k \equiv p_k) \wedge y_1 \wedge ... \wedge y_m$ with some $y_1, ..., y_m \in \text{vars}(\phi_{\text{xor}}) \setminus \{x_1, ..., x_k\}$, a vertex v with $L(v) = (\bot \equiv \top)$ in it, and a cut W for v such that $\mathit{Expl}_{\oplus}(v, W) + L(v) = (x'_1 \oplus ... \oplus x'_l \equiv p')$ for some $\{x'_1, ..., x'_l\} \subseteq \{x_1, ..., x_k\}$ and $p' \in \{\bot, \top\}$ such that $\phi_{\text{xor}} \models (x'_1 \oplus ... \oplus x'_l \equiv p')$.*

Implying clauses for xor-implied literals provided by Gauss-Jordan elimination xor-deduction system presented in Section 3.4 are based on prime implicate xor-constraints because Gauss-Jordan elimination uses reduced row-echelon form matrices and explanations are derived from the rows of such matrices. This has an interesting consequence; provided that unlimited restarts are allowed, the CDCL SAT solver equipped with UP xor-deduction system, parity explanations and xor-constraint learning as described in Section 4.2 can in theory simulate Gauss-Jordan elimination xor-deduction system in the DPLL(XOR) framework on instances where each variable occurs at most three times. All linear combinations that the Gauss-Jordan xor-deduction system needs in its derivations can be learned first with parity explanations.

7.4 Simulating Equivalence Reasoning with Unit Propagation

The connection between equivalence reasoning and xor-cycles studied in Section 5.2 enables us to consider a potentially more efficient way to implement equivalence reasoning. We now present three translations that add redundant xor-constraints with the aim that unit propagation is enough to always deduce all xor-implied literals in the resulting xor-constraint conjunction. The first translation is based on the xor-cycles of the formula and does not add auxiliary variables, the second translation is based on explicitly communicating equivalences between the variables of the original formula using auxiliary variables, and the third translation combines the first two.

The redundant xor-constraint conjunction, called an EC-*simulation formula* ψ, added to ϕ_{xor} by a translation should satisfy the following: (i) the satisfying truth assignments of ϕ_{xor} are exactly the ones of $\phi_{\text{xor}} \wedge \psi$ when projected to $\text{vars}(\phi_{\text{xor}})$, and (ii) if $\hat{l}$ is EC-derivable from $\phi_{\text{xor}} \wedge (l_1) \wedge \cdots \wedge (l_k)$, then $\hat{l}$ is UP-derivable from $(\phi_{\text{xor}} \wedge \psi) \wedge (l_1) \wedge \cdots \wedge (l_k)$.

7.4.1 Simulation without extra variables

We first present an EC-simulation formula for a given 3-xor normal form xor-constraint conjunction ϕ_{xor} without introducing additional variables. The translation adds one xor-constraint with the all outer variables per xor-cycle:

$$\mathit{cycles}(\phi_{\text{xor}}) = \bigwedge_{XC(\langle x_1,\ldots,x_n\rangle,\langle y_1,\ldots,y_n\rangle,p)\subseteq\phi_{\text{xor}}} (y_1 \oplus \ldots \oplus y_n \equiv p)$$

For example, for the conjunction ϕ_{xor} in Figure 5.5 $\mathit{cycles}(\phi_{\text{xor}}) = (a \oplus e \oplus j \equiv \bot)$.

Theorem 17. *If ϕ_{xor} is a 3-xor normal form xor-constraint conjunction, then $\mathit{cycles}(\phi_{\text{xor}})$ is an* EC*-simulation formula for ϕ_{xor}.*

The translation is intuitively suitable for problems that have a small number of xor-cycles, such as the DES cipher. Each instance of our DES benchmark (4 rounds, 2 blocks) has 28–32 xor-cycles. We evaluated the translation experimentally on this benchmark using cryptominisat 2.9.2, minisat 2.0, minisat 2.2, and minisat 2.0 extended with the UP xor-reasoning module. The benchmark set has 51 instances and the clauses of each instance are permuted 21 times randomly to negate the effect of propagation order. The results are shown in Figure 7.3. The translation manages to slightly reduce solving time for cryptominisat, but this does not happen for other solver configurations based on minisat, so the slightly improved performance is not completely due to simulation of equivalence reasoning using unit propagation. The xor-part (320 xor-constraints of which 192 tree-like) in DES is negligible compared to CNF-part (over 28000 clauses), so a great reduction in solving time is not expected.

Although equivalence reasoning can be simulated with unit propagation by adding an xor-constraint for each xor-cycle, this is not feasible for all instances in practice due to the large number of xor-cycles. We now show that, if auxiliary variables are not allowed, there are families of xor-constraint conjunctions without polynomial-size EC-simulation formulas. Consider the xor-constraint conjunction $D(n) = (x_1 \oplus x_{n+1} \oplus y) \wedge \bigwedge_{i=1}^{n} (x_i \oplus x_{i,a} \oplus x_{i,b}) \wedge (x_{i,b} \oplus x_{i,c} \oplus x_{i+1}) \wedge (x_i \oplus x_{i,d} \oplus x_{i,e}) \wedge (x_{i,e} \oplus x_{i,f} \oplus x_{i+1})$ whose constraint graph is shown in Figure 7.4. Observe that $D(n)$ is cycle-partitionable and thus Subst/EC-deducible. But all its EC-simulation formulas are at least of exponential size if no auxiliary variables are allowed:

Lemma 18 (Lem. 2 of Laitinen *et al.* [2014a])**.** *Any* EC*-simulation formula ψ for $D(n)$ with* $\text{vars}(\psi) = \text{vars}(D(n))$ *contains at least 2^n xor-constraints.*

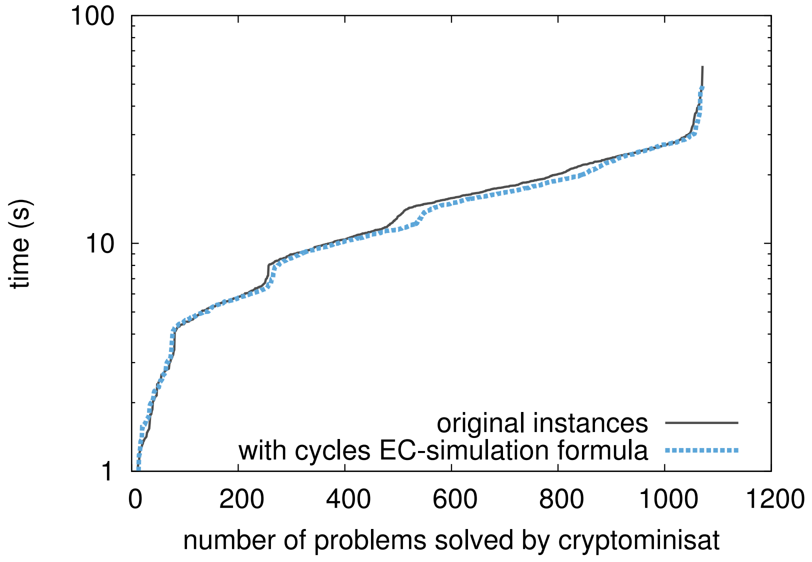


Figure 7.3. Solving time with and without $cycles(\phi_{\text{xor}})$ translation on DES instances

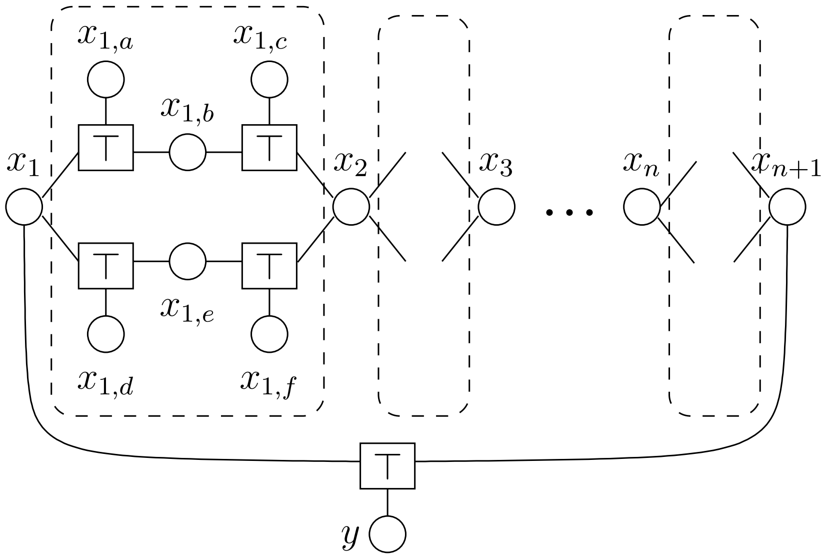


Figure 7.4. The constraint graph of $D(n)$

7.4.2 Simulation with extra variables: basic version

Our second translation $\mathrm{Eq}(\phi_{\mathrm{xor}})$ avoids the exponential increase in size by introducing a quadratic number of auxiliary variables.

Now the translation is

$$\mathrm{Eq}(\phi_{\mathrm{xor}}) \;=\; \left(\bigwedge_{(x_i \oplus x_j \oplus x_k \equiv p) \in \phi_{\mathrm{xor}}} \begin{array}{c} (e_{ij} \oplus x_k \equiv p \oplus \top) \wedge (e_{ik} \oplus x_j \equiv p \oplus \top) \wedge \\ (x_i \oplus e_{jk} \equiv p \oplus \top) \end{array} \right) \wedge \left(\bigwedge_{x_i, x_j, x_k \in \mathrm{vars}(\phi_{\mathrm{xor}}), i<j<k} (e_{ij} \oplus e_{jk} \oplus e_{ik} \equiv \top) \right)$$

where (i) the first line ensures that if we can deduce that two variables in a ternary xor-constraint are (in)equivalent, then we can deduce the value of the third variable, and vice versa, and (ii) the second line encodes transitivity of (in)equivalences. The translation enables unit propagation to deduce all EC-derivable literals over the variables in the original xor-constraint conjunction:

Theorem 19. *If* ϕ_{xor} *is an xor-constraint conjunction in 3-xor normal form, then* $\mathrm{Eq}(\phi_{\mathrm{xor}})$ *is an* EC*-simulation formula for* ϕ_{xor}.

7.4.3 Simulation with extra variables: optimized version

The translation $\mathrm{Eq}(\phi_{\mathrm{xor}})$ adds a cubic number of xor-constraints with respect to the variables in ϕ_{xor}. This is infeasible for many real-world instances. The third translation combines the first two translations by implicitly taking into account the xor-cycles in ϕ_{xor} while adding auxiliary variables where needed. The translation $\mathrm{Eq}^{\star}(\phi_{\mathrm{xor}})$ is presented in Figure 7.5. The xor-constraints added by $\mathrm{Eq}^{\star}(\phi_{\mathrm{xor}})$ are a subset of $\mathrm{Eq}(\phi_{\mathrm{xor}})$ and the meaning of the variable e_{ij} remains the same. The intuition behind the translation, on the level of constraint graphs, is to iteratively shorten xor-cycles by "eliminating" one variable at a time by adding auxiliary variables that "bridge" possible equivalences over the eliminated variable. The line 2 in the pseudo-code picks a variable x_j to eliminate. While the correctness of the translation does not depend on the choice, we decided to use a heuristic approach to pick a variable that shares xor-constraints with fewest variables because the number of xor-constraints produced in lines 3–9 is then smallest. The loop in line 3 iterates over all possible xor-cycles where the selected variable x_j and two "neighboring" non-eliminated variables x_i,x_k may occur as inner variables. The line 4

```
Eq$^\star(\phi_{\text{xor}})$: start with $\phi'_{\text{xor}} = \phi_{\text{xor}}$ and $V = \text{vars}(\phi_{\text{xor}})$
1.  while $(V \neq \emptyset)$:
2.      $x_j \leftarrow$ extract a variable $v$ from $V$
minimizing $|\,\text{vars}(\{C \in \phi'_{\text{xor}} \mid v \in \text{vars}(C)\}) \cap V|$
3.      for each $(x_i \oplus x_j \oplus e_{ij} \equiv p_{ij}), (x_j \oplus x_k \oplus e_{jk} \equiv p_{jk}) \in \phi'_{\text{xor}}$
                        such that $x_i, x_k \in V \wedge x_i \neq x_j \neq x_k$
4.          if $(x_i \oplus x_k \oplus y \equiv p'_{ik}) \in \phi'_{\text{xor}}$
5.              $e_{ik} \leftarrow y$; $p_{ik} \leftarrow p'_{ik}$
6.          else
7.              $e_{ik} \leftarrow$ new variable; $p_{ik} \leftarrow \top$
8.              $\phi'_{\text{xor}} \leftarrow \phi'_{\text{xor}} \wedge (x_i \oplus x_k \oplus e_{ik} \equiv p_{ik})$
9.          $\phi'_{\text{xor}} \leftarrow \phi'_{\text{xor}} \wedge (e_{ij} \oplus e_{jk} \oplus e_{ik} \equiv p_{ij} \oplus p_{jk} \oplus p_{ik})$
10. return $\phi'_{\text{xor}} \backslash \phi_{\text{xor}}$
```

Figure 7.5. The Eq* translation

checks if there already is an xor-constraint that has both x_i and x_k. If so, then in line 5 an existing variable is used as e_{ik} capturing the equivalence between the variables x_i and x_k. If the variable p_{ik} is $\top$, then e_{ik} is true when the variables x_i and x_k have the same value. The line 9 adds an xor-constraint ensuring that transitivity of equivalences between the variables x_i, x_j, and x_k can be handled by unit propagation.

Example 13. *Consider the xor-constraint conjunction* $\phi_{\text{xor}} = (x_1 \oplus x_2 \oplus x_4 \equiv \top) \wedge (x_2 \oplus x_3 \oplus x_5 \equiv \top) \wedge (x_5 \oplus x_7 \oplus x_8 \equiv \top) \wedge (x_4 \oplus x_6 \oplus x_7 \equiv \top)$ *shown in Figure 7.6(a). The translation* Eq$^\star(\phi_{\text{xor}})$ *first selects the variables in* $\{x_1, x_3, x_6, x_8\}$ *one by one as each appears in only one xor-constraint. The loop in lines 3–9 is not executed for any of them. The remaining variables are* $V = \{x_2, x_4, x_5, x_7\}$. *Assume that* x_2 *is selected. The loop in lines 3–9 is entered with values* $x_i{=}x_4$, $x_j{=}x_2$, $e_{ij}{=}x_1$, $x_k{=}x_5$, $e_{jk}{=}x_3$, $p_{ij}{=}\top$, *and* $p_{jk}{=}\top$. *The condition in line 4 fails, so the xor-constraints* $(x_4 \oplus x_5 \oplus e_{45} {\equiv} \top)$ *and* $(x_1 \oplus x_3 \oplus e_{45} {\equiv} \top)$,*where* e_{45} *is a new variable, are added. The resulting instance is shown in Figure 7.6(b). Assume that* x_5 *is selected. The loop in lines 3–9 is entered with values* $x_i{=}x_4$, $x_j{=}x_5$, $e_{ij}{=}e_{45}$, $x_k{=}x_7$, $e_{jk}{=}x_8$, $p_{ij}{=}\top$, *and* $p_{jk}{=}\top$. *The condition in line 4 is true, so* $e_{ik}{=}x_6$, *and the xor-constraint* $(x_6 \oplus x_8 \oplus e_{45} {\equiv} \top)$ *is added in line 9. The final result is shown in Figure 7.6(c).*

Theorem 20. *If* ϕ_{xor} *is an xor-constraint conjunction in 3-xor normal form, then* Eq$^\star(\phi_{\text{xor}})$ *is an* EC*-simulation formula for* ϕ_{xor}.

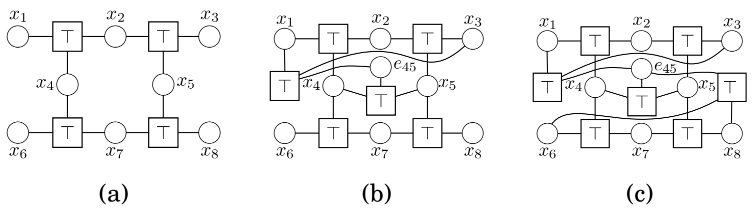


Figure 7.6. Constraint graphs illustrating how the translation Eq* adds new xor-constraints

7.5 Simulating Stronger Parity Reasoning with Unit Propagation

The previous section presents three different translations for simulating equivalence reasoning with unit propagation. We now present a translation that adds redundant xor-constraints and auxiliary variables in the problem guaranteeing that unit propagation is enough to always deduce all xor-implied literals in the resulting xor-constraint conjunction. The translation thus effectively simulates a complete parity reasoning engine based on incremental Gauss-Jordan elimination presented in [IV] and in Han and Jiang [2012]. The translation is based on ensuring that each relevant linear combination of original variables has a corresponding "alias" variable, and adding xor-constraints that enable unit propagation to infer values of "alias" variables when corresponding linear combinations are implied. The translation, which is exponential in the worst case, can be made polynomial by bounding the length of linear combinations to consider. While unit propagation may not be able then to deduce all xor-implied literals, the overall performance may be improved greatly.

The redundant xor-constraint conjunction, called a *GE-simulation formula* ψ, added to ϕ_{xor} by the translation should satisfy the following: (i) the satisfying truth assignments of ϕ_{xor} are exactly the ones of $\phi_{\text{xor}} \wedge \psi$ when projected to $\text{vars}(\phi_{\text{xor}})$, and (ii) if ϕ_{xor} is satisfiable and $\phi_{\text{xor}} \wedge l_1 \wedge \cdots \wedge l_k \models \hat{l}$, then $\hat{l}$ is UP-derivable from $(\phi_{\text{xor}} \wedge \psi) \wedge l_1 \wedge \cdots \wedge l_k$, and (iii) if ϕ_{xor} is unsatisfiable, then $(\phi_{\text{xor}} \wedge \psi) \vdash_{\text{UP}} (\bot \equiv \top)$.

The translation k-Ge, presented in Figure 7.8, where k stands for the maximum length of linear combination to consider, "eliminates" each variable of the conjunction ϕ_{xor} at a time and adds xor-constraints produced by the subroutine translation ptable, presented in Figure 7.7. Although the choice of variable to eliminate does not affect the correctness of the translation, we employ a greedy heuristic to select a variable that shares xor-constraints with the fewest variables. The number of xor-constraints produced in the subroutine ptable is then the smallest. The translation

```
ptable($Y, \phi_{\text{xor}}, k$): start with $\phi'_{\text{xor}} = \phi_{\text{xor}}$
1.  for each $Y' \subseteq Y$ such that $|Y'| \leq k$ and $Y' \neq \emptyset$
2.      if there is no $a \in \text{vars}(\phi'_{\text{xor}})$ such that $(a \oplus Y' \equiv \bot)$ is in $\phi'_{\text{xor}}$
3.          $\phi'_{\text{xor}} \leftarrow \phi'_{\text{xor}} \wedge (a \oplus Y' \equiv \bot)$ where $a$ is a new “alias” variable for $Y'$
4.      if $(Y' \equiv p)$ is in $\phi'_{\text{xor}}$ and $(a \equiv p)$ is not in $\phi'_{\text{xor}}$
5.          $\phi'_{\text{xor}} \leftarrow \phi'_{\text{xor}} \wedge (a \equiv p)$
6.  for each pair of subsets $Y_1, Y_2 \subseteq Y$ such that $|Y_1| \leq k$, $|Y_2| \leq k$, and $Y_1 \neq Y_2$
7.      if there is “alias” variable $a_3 \in \text{vars}(\phi'_{\text{xor}})$
                        such that $(a_3 \oplus (Y_1 \oplus Y_2) \equiv \bot)$ is in $\phi'_{\text{xor}}$
8.          $a_1 \leftarrow$ the “alias” variable $v$ such that $(v \oplus Y_1 \equiv \bot)$ is in $\phi'_{\text{xor}}$
9.          $a_2 \leftarrow$ the “alias” variable $v$ such that $(v \oplus Y_2 \equiv \bot)$ is in $\phi'_{\text{xor}}$
10.         if $(a_1 \oplus a_2 \oplus a_3 \equiv \bot)$ is not in $\phi'_{\text{xor}}$
11.             $\phi'_{\text{xor}} \leftarrow \phi'_{\text{xor}} \wedge (a_1 \oplus a_2 \oplus a_3 \equiv \bot)$
12. return $\phi'_{\text{xor}} \setminus \phi_{\text{xor}}$
```

Figure 7.7. The ptable translation

```
$k$-Ge($\phi_{\text{xor}}$): start with $\phi'_{\text{xor}} = \phi_{\text{xor}}$ and $V = \text{vars}(\phi_{\text{xor}})$
1.  while ($V \neq \emptyset$):
2.      Let $\text{clauses}(x, \phi'_{\text{xor}}) = \{D \mid D$ in $\phi'_{\text{xor}}$ and $x \in \text{vars}(D)\}$
3.      Let $x$ be a variable in $V$ minimizing $|\text{vars}(\text{clauses}(x, \phi'_{\text{xor}})) \cap V|$
4.      $\phi'_{\text{xor}} \leftarrow \phi'_{\text{xor}} \wedge \text{ptable}(\text{vars}(\text{clauses}(x, \phi'_{\text{xor}})) \cap V, \phi'_{\text{xor}}, k)$
5.      Remove $x$ from $V$
6.  return $\phi'_{\text{xor}} \setminus \phi_{\text{xor}}$
```

Figure 7.8. The k-Ge translation

$\text{ptable}(Y, \psi, k)$ adds “alias” variables and $O(2^{2k}) + |\phi_{\text{xor}}|$ xor-constraints to ψ with the aim to simulate Gauss-Jordan row operations involving at most k variables in the xor-constraints of the eliminated variable (the set Y) and no other variables. Provided that the maximum length of linear combinations to consider, the parameter k, is high enough ($k \geq |Y|$), the resulting xor-constraint conjunction $\psi \wedge \text{ptable}(Y, \psi, k)$ has a UP-*propagation table* for the set of variables $Y \subseteq \text{vars}(\phi_{\text{xor}})$, denoted by $Y \subseteq_{\text{UP}} \psi$, meaning that the following conditions hold for all $Y', Y_1, Y_2 \subseteq Y$:

PT1: There is an “alias” variable for every non-empty subset of Y: if Y' is a non-empty subset of Y, then there is a variable $a \in \text{vars}(\psi)$ such that $(a \oplus Y' \equiv \bot)$ is in ψ, where $(a \oplus Y' \equiv \bot)$ for $Y' = \{y'_1, \ldots, y'_n\}$ means $(a \oplus y'_1 \oplus \cdots \oplus y'_n \equiv \bot)$.

PT2: There is an xor-constraint for propagating the symmetric difference of any two subsets of Y: if $Y_1 \subseteq Y$ and $Y_2 \subseteq Y$, then there are variables $a_1, a_2, a_3 \in \mathrm{vars}(\psi)$ such that $(a_1 \oplus Y_1 \equiv \bot), (a_2 \oplus Y_2 \equiv \bot), (a_3 \oplus (Y_1 \oplus Y_2) \equiv \bot)$, and $(a_1 \oplus a_2 \oplus a_3 \equiv \bot)$ are in ψ.

PT3: Alias variables of original xor-constraints having only variables of Y are assigned: if $(Y' \equiv p)$ is an xor-constraint in ψ such that $Y' \subseteq Y$, then there is a variable $a \in \mathrm{vars}(\psi)$ such that $(a \oplus Y' \equiv \bot)$ and $(a \equiv p)$ is in ψ.

A UP-propagation table for a set of variables Y in ψ guarantees that if some alias variables $a_1, \ldots, a_n \in \mathrm{vars}(\psi)$ binding the variable sets $Y_1, \ldots, Y_n \subseteq Y$ are assigned, the alias variable $a \in \mathrm{vars}(\psi)$ bound to the linear combination $(Y_1 \oplus \cdots \oplus Y_n)$ is UP-deducible: $\psi \wedge (a_1 \equiv p_1) \wedge \cdots \wedge (a_n \equiv p_n) \vdash_{\mathsf{UP}} (a \equiv p_1 \oplus \cdots \oplus p_n)$.

Provided that sufficiently long linear combinations are considered (the parameter k), UP-propagation tables added by the k-Ge enable unit propagation to always deduce all xor-implied literals, and thus simulate a complete Gauss-Jordan propagation engine:

Theorem 21. *If ϕ_{xor} is an xor-constraint conjunction, then k-$\mathrm{Ge}(\phi_{\text{xor}})$ is a GE-simulation formula for ϕ_{xor} provided that $k = |\mathrm{vars}(\phi_{\text{xor}})|$.*

Example 14. *Consider the xor-constraint conjunction $\phi^{(0)}_{\text{xor}} = (x_1 \oplus x_6 \oplus x_7 \equiv \top) \wedge (x_2 \oplus x_3 \oplus x_7 \equiv \top) \wedge (x_2 \oplus x_5 \oplus x_8 \equiv \bot) \wedge (x_3 \oplus x_4 \oplus x_5 \equiv \top) \wedge (x_4 \oplus x_6 \oplus x_8 \equiv \bot)$ illustrated in Figure 6.10.*

With the elimination order $(x_1, x_7, x_4, x_5, x_2, x_3, x_6, x_8)$ and $k = 4$, the translation k-Ge first extends ϕ_{xor} to $\phi^{(1)}_{\text{xor}}$ with $\mathrm{ptable}(\{x_1, x_6, x_7\}, \phi_{\text{xor}}, k)$. These xor-constraints include (i) the "alias binding constraints" (see PT1) $a_1 \oplus x_1 \equiv \bot$, $a_{6,7} \oplus x_6 \oplus x_7 \equiv \bot$, $a_{1,6,7} \oplus x_1 \oplus x_6 \oplus x_7 \equiv \bot$, (ii) the "linear combination constraint" (see PT2) $a_1 \oplus a_{6,7} \oplus a_{1,6,7} \equiv \bot$, and (iii) the "original constraint binder" (see PT3) $a_{1,6,7} \equiv \top$, where $a_{i,\ldots}$ is the alias for the subset $\{x_i, \ldots\}$ of the original variables. After unit propagation, these constraints imply the binary constraint $a_1 \oplus a_{6,7} \equiv \top$ allowing us to deduce x_1 from the parity $a_{6,7}$ of x_6 and x_7.

Eliminating x_4 adds $\mathrm{ptable}(\{x_3, x_4, x_5, x_6, x_8\}, \phi^{(2)}_{\text{xor}}, k)$, including the constraints $a_{3,4,5} \oplus a_{4,6,8} \oplus a_{3,5,6,8} \equiv \bot$, $a_{3,4,5} \equiv \top$, and $a_{4,6,8} \equiv \top$, propagating $a_{3,5,6,8} \equiv \top$.

Eliminating x_5 adds $\mathrm{ptable}(\{x_2, x_3, x_5, x_6, x_8\}, \phi^{(3)}_{\text{xor}}, k)$ (observe that x_6 is in the set as it occurs in the constraint $a_{3,5,6,8} \oplus x_3 \oplus x_5 \oplus x_6 \oplus x_8 \equiv \bot$ added in the previous step), including $a_{2,5,8} \oplus a_{2,3,6} \oplus a_{3,5,6,8} \equiv \bot$ and $a_{2,5,8} \equiv \bot$.

At this point we could already unit propagate $x_1 \equiv \top$ *(from* $a_{3,5,6,8} \equiv \top$*,* $a_{2,5,8} \equiv \bot$*, and* $a_{2,5,8} \oplus a_{2,3,6} \oplus a_{3,5,6,8} \equiv \bot$ *we get* $a_{2,3,6} \equiv \top$ *and from this then* $a_{6,7} \equiv \bot$ *and finally* $a_1 \equiv \top$*, i.e.* $x_1 \equiv \top$*).*

Note that the translation 3-Ge(ϕ_{xor}) *is not a GE-simulation formula for* ϕ_{xor} *because* ptable *does not add "alias" variables for any 4-subset of original variables and the linear combination of any two original xor-constraints has at least four variables.*

7.5.1 Propagation-preserving xor-simplification

Some of the xor-constraints added by k-Ge can be redundant regarding unit propagation. We now present a simplification method that preserves literals that can be implied by unit propagation. There are two simplification rules, given a pair of xor-constraint conjunctions $\langle \phi_a, \phi_b \rangle$ (initially $\langle \phi_{\text{xor}}, \emptyset \rangle$): [S1] an xor-constraint D in ϕ_a be can moved to ϕ_b, resulting in $\langle \phi_a \setminus \{D\}, \phi_b \cup \{D\} \rangle$, and [S2] an xor-constraint D in ϕ_a can be simplified with an xor-constraint D' in ϕ_b to $(D + D')$ provided that $|\text{vars}(D') \cap \text{vars}(D)| \geq |\text{vars}(D')| - 1$, resulting in $\langle (\phi_a \setminus \{D\}) \cup \{D + D'\}, \phi_b \rangle$.

Theorem 22. *If* $\langle \phi'_a, \phi'_b \rangle$ *is the result of applying one of the simplification rules to* $\langle \phi_a, \phi_b \rangle$ *and* $\phi_a \wedge \phi_b \wedge l_1 \wedge \cdots \wedge l_k \vdash_{\mathsf{UP}} \hat{l}$*, then* $\phi'_a \wedge \phi'_b \wedge l_1 \wedge \cdots \wedge l_k \vdash_{\mathsf{UP}} \hat{l}$*.*

Example 15. *The conjunction* 3-Ge($(x_1 \oplus x_2 \oplus x_3 \oplus x_4 \equiv \bot)$) *contains the alias binding constraints* $D_1 := (a_{1,2,3,4} \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \equiv \bot)$*,* $D_2 := (a_{1,2} \oplus x_1 \oplus x_2 \equiv \bot)$*,* $D_3 := (a_{3,4} \oplus x_3 \oplus x_4 \equiv \bot)$*, as well as the linear combination constraint* $D_4 := (a_{1,2} \oplus a_{3,4} \oplus a_{1,2,3,4} \equiv \bot)$*. The alias binding constraint* D_1 *can in fact be eliminated by first applying the rule S1 to the xor-constraints* D_2*,* D_3*, and* D_4*. Then, by using the rule S2, the xor-constraint* D_1 *is simplified first with* D_2 *to* $(a_{1,2,3,4} \oplus a_{1,2} \oplus x_3 \oplus x_4 \equiv \bot)$ *and then with* D_3 *to* $(a_{1,2,3,4} \oplus a_{1,2} \oplus a_{3,4} \equiv \bot)$*, and finally with* D_4 *to* $(\bot \equiv \bot)$*.*

7.5.2 Experimental Evaluation

To evaluate the translation k-Ge, we ran cryptominisat 2.9.6, glucose 2.3, and zenn 0.1.0 on the seven benchmark families with the translations k-Ge and Eq*[1]. It is intractable to simulate full Gauss-Jordan elimination for these instances, so we adjusted the k-value of each call to the subroutine ptable(Y, ψ, k) to limit the number of additional xor-constraints.

[1]The experimental evaluation of Eq* and k-Ge in [V] is partially erroneous. See Laitinen *et al.* [2013] for the corrected version.

The translation was computed for each connected component separately. We found relatively good performance when solving SAT Competition instances by (i) stopping when $|Y| > 66$, (ii) setting $k = 1$ when it was detected that unit propagation deduces all xor-implied literals, (iii) setting $k = 2$ when $|Y| \in [10, 66]$ or when $|Y| < 10$ and it was detected that equivalence reasoning deduces all xor-implied literals, (iv) setting $k = 3$ when $|Y| \in [6, 9]$, setting $k = |Y|$ when $|Y| \leq 5$. With these parameters, the worst-case number of xor-constraints added by the subroutine ptable is 2145. Figure 7.9 shows the increase in formula size by the translation k-Ge on SAT Competition instances. The translation Eq*was computed in a similar way. The results, shown in Figures 7.10, 7.11, 7.12, 7.22, 7.14, 7.15, 7.16, 7.17, 7.18, 7.19, 7.20, and 7.21, were obtained by running each solver configuration on each benchmark instance for at most one hour on 20-core Intel E5-2680 v2 with 256 GB RAM per processor. Memory limit for one solver instance was set to 10 GB. The solver configuration k-Ge computes the translation k-Ge before running the corresponding solver, and k-Ge simp additionally preprocesses the instance with propagation-preserving xor-simplification. The solver configurations Eq* and Eq* simp behave similarly but use the translation Eq*. The time spent in computing the translations is included.

When solving with cryptominisat, the translations reduce solving time on A5/1 instances. The translations reduce number of decisions on A5/1, FEAL, Trivium, and some SAT Competition instances. The translation k-Ge reduces number of decisions more than Eq* on FEAL and Trivium instances.

When solving with glucose, the translations reduce solving time for FEAL instances. The translations reduce number of decisions on A5/1, FEAL, Trivium and some SAT Competition instances. The translation k-Ge reduces number of decisions more than Eq* on Trivium instances.

When solving with zenn, the translations enable the solver to solve more A5/1 instances. The translations reduce number of decisions on FEAL, Trivium, and SAT Competition instances.

Propagation-preserving xor-simplification incurs a noticeable delay proportional to number of xor-constraints before search for solution can be started. The translation Eq* may be more compact compared to the translation k-Ge because it does not benefit from the xor-simplification. The translation k-Ge, however, performs better with the xor-simplification on i) with cryptominisat, on A5/1 and Trivium instances, ii) with glucose and

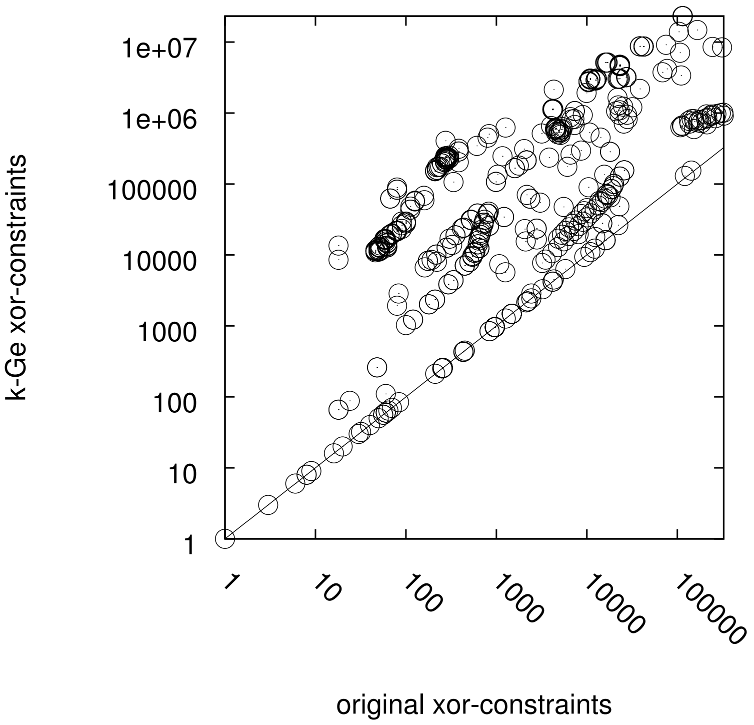


Figure 7.9. Xor-constraints in SAT 05-11 instances

zenn, on A5/1 and Hitag2 instances.

The choice of parameters for the translations was made based on the performance on one benchmark family only and the same parameters were used for all benchmark families. It is left for future work to find out whether it is possible to choose more appropriate parameters for other benchmark families.

Finally, we present an overall comparison of the solver configurations presented in Chapters 3, 4, and 7. Figures 7.24, 7.25, 7.26, 7.27, 7.28, 7.29, and 7.30 show the number of solved instances, the median number of decisions and the median solving time for the 24 solver configurations on the seven benchmark families.

7.5.3 Polynomial-size translation for instances of bounded treewidth

The number of xor-constraints produced by the translation k-Ge depends strongly on the instance, as shown in Figure 7.9. Now we connect the worst-case size of a ptable-based GE-simulation formula to treewidth, a well-known structural property of (constraint) graphs used often to characterize the hardness of solving a problem, e.g. an instance of CSP with bounded treewidth can be solved in polynomial time (shown by Freuder [1990]). See Samer and Szeider [2010] for a more complete overview on solving CSPs with bounded treewidth. They use the term *incidence graph* to discuss constraint graphs. We apply the decomposition method presented in Section 6.3 to produce a polynomial-size GE-simulation formula for instances of bounded treewidth. We also present some found upper bounds for treewidth in SAT Competition instances that illustrate to what extent parity reasoning can be simulated through unit propagation. As illustrated in Section 6.3, each pair of adjacent nodes in a tree decomposition defines a cut variable set, so it suffices to add a UP-propagation table for each node's variable set. If an xor-constraint conjunction has a bounded treewidth, the tree decomposition can be used to construct a polynomial-size GE-simulation formula:

Theorem 23. *If $\{X_1, \dots, X_n\}$ is the family of variable sets in the tree decomposition of the primal graph of an xor-constraint conjunction ϕ_{xor} and $\phi_0, \dots, \phi_n$ is a sequence of xor-constraint conjunctions such that $\phi_0 = \phi_{\text{xor}}$ and $\phi_i = \phi_{i-1} \wedge \text{ptable}(X_i, \phi_{i-1}, |X_i|)$ for $i \in \{1, \dots, n\}$, then $\phi_n \setminus \phi_{\text{xor}}$ is a GE-simulation formula for ϕ_{xor} with $O(n2^{2k}) + |\phi_{\text{xor}}|$ xor-constraints, where $k = \max(|X_1|, \dots, |X_n|)$.*

To find out to what extent unit propagation can simulate stronger parity reasoning, we studied the 474 SAT Competition benchmark instances. We applied the junction tree algorithm described in Pearl [1982] to get an upper bound for treewidth. The found treewidths are shown in Figure 7.23. There are some instances that have compact GE-simulation formulas, but for the majority of the instances, full GE-simulation formula is likely to be intractably large. For these instances a powerful solution technique can be to choose a suitable propagation method for each biconnected component separately, either through a translation or an xor-reasoning module.

Related work. Subst-simulation and GE-simulation formulas are closely related to the problem of finding good CNF-representations of systems of linear equations over the two-element field (conjunctions of xor-constraints) studied in Gwynne and Kullmann [2013]. In their work, the basic quality criterion is "arc consistency", that is, for every partial truth assignment to the (original) variables of the conjunction of xor-constraints, all implied literals can be deduced by unit propagation. They show that there is no AC-representation of polynomial size for arbitrary conjunction of xor-constraints. They develop a translation that produces an AC-representation of a conjunction of xor-constraints by enumerating all linear combinations of the xor-constraints, and then introducing auxiliary variables to split longer xor-constraints to an equisatisfiable conjunction of xor-constraints of at most three variables. The translation adds at most 2^m new xor-constraints where m is the number of original xor-constraints, and it is thus more compact than the translation k-Ge, which is exponential in the number of variables. However, in practice, the translation k-Ge can produce much smaller GE-simulation formulas because it takes into account the structure of the xor-constraint conjunction.

To obtain CNF-representations stronger than mere AC, Gwynne and Kullmann [2013] consider the class $\mathcal{PC}$ which stands for propagation-complete clause-sets, introduced in Bordeaux and Marques-Silva [2012]. The stronger criterion requires that for all partial truth assignments to the variables of a conjunction of xor-constraints, involving also possible auxiliary variables, all implied literals can be deduced by unit propagation. Gwynne and Kullmann [2013] develop a propagation-completeness ensuring translation for xor-constraint conjunctions of at most two xor-constraints. They define an *incidence graph* as bipartite graph where i) each variable has a node and each xor-constraint has a node, and ii) there is an edge between a variable node and an xor-constraint node if the variable has an occurrence in the xor-constraint. They conjecture in Conjecture 11.1 that there is a propagation-completeness ensuring translation that the number of added xor-constraints is exponential in the treewidth of the incidence graph of the original conjunction of xor-constraints. This is closely related to Theorem where we define a polynomial GE-simulation formula for xor-constraint conjunctions whose primal graphs have bounded treewidth. We conjecture that the translation also ensures propagation-completeness.

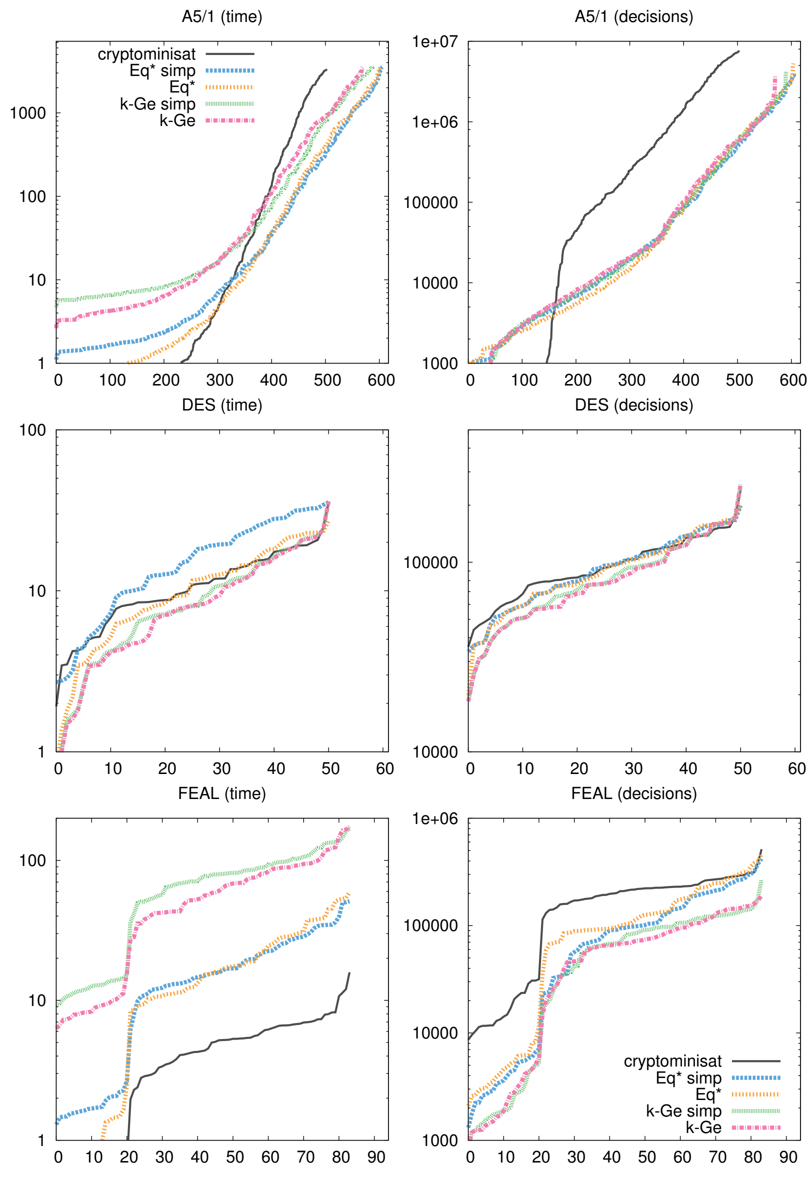


Figure 7.10. Solving time and number of decisions as functions of solved instances (cryptominisat part 1/3)

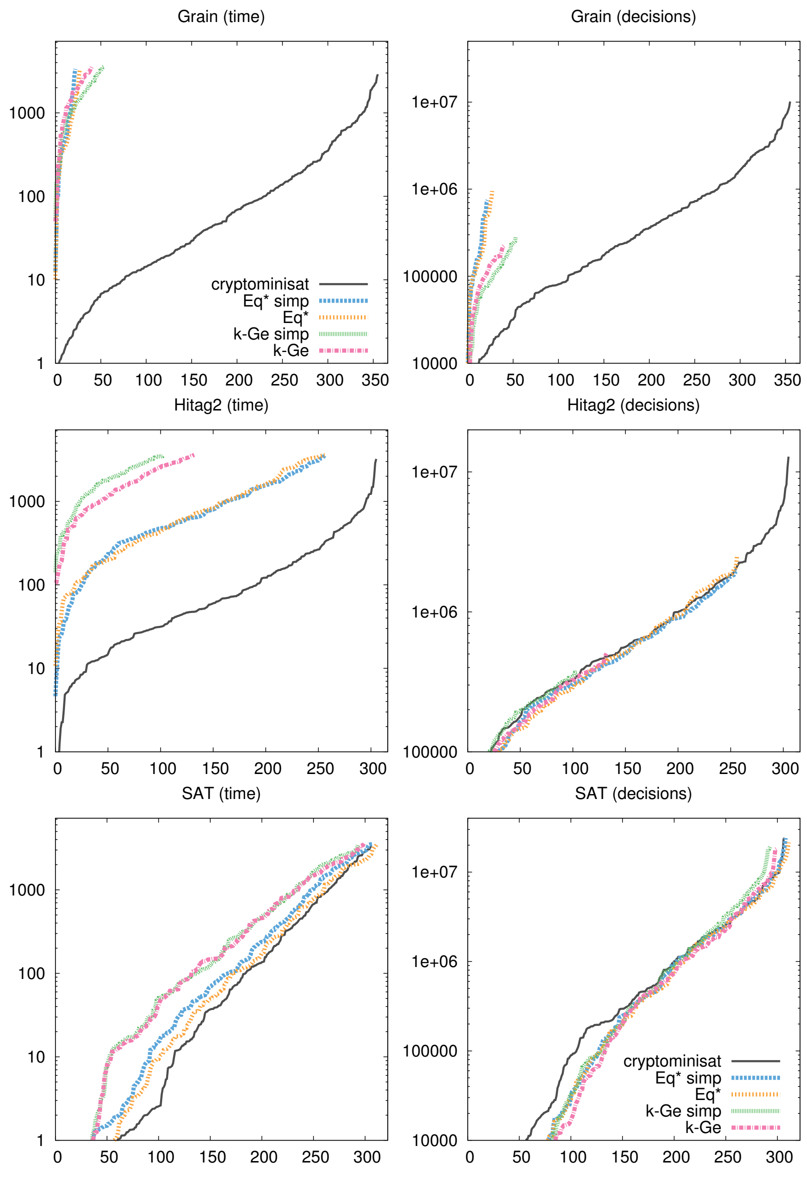


Figure 7.11. Solving time and number of decisions as functions of solved instances (cryptominisat part 2/3)

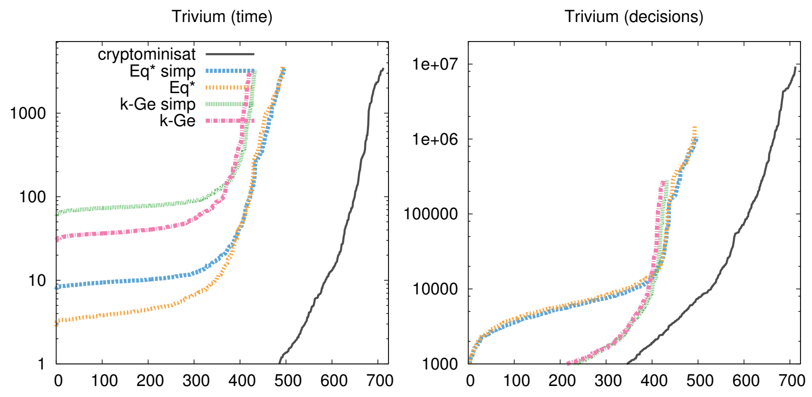


Figure 7.12. Solving time and number of decisions as functions of solved instances (cryptominisat part 3/3)

A5/1 (640 instances)			
Solver	#	Decisions	Time (s)
Eq* simp	607	24722	9.0
Eq*	605	19724	6.8
k-Ge simp	591	23558	21.4
k-Ge	571	24739	22.3
cryptominisat	504	340275	8.3

DES (51 instances)			
Solver	#	Decisions	Time (s)
k-Ge	51	76706	8.1
k-Ge simp	51	77799	8.0
Eq*	51	92624	11.6
cryptominisat	51	93695	10.9
Eq* simp	51	96072	16.6

FEAL (84 instances)			
Solver	#	Decisions	Time (s)
k-Ge	84	66274	54.8
k-Ge simp	84	68468	76.0
Eq* simp	84	91303	14.9
Eq*	84	95753	14.8
cryptominisat	84	200137	4.4

Grain (357 instances)			
Solver	#	Decisions	Time (s)
cryptominisat	356	253777	46.2
k-Ge simp	54	-	-
k-Ge	41	-	-
Eq*	28	-	-
Eq* simp	23	-	-

Hitag2 (306 instances)			
Solver	#	Decisions	Time (s)
cryptominisat	306	577225	61.9
Eq* simp	257	519855	831.0
Eq*	257	531972	852.6
k-Ge	133	-	-
k-Ge simp	105	-	-

SAT (474 instances)			
Solver	#	Decisions	Time (s)
Eq*	312	2092268	588.6
Eq* simp	309	1961380	642.4
cryptominisat	308	2090916	442.0
k-Ge	300	1754659	1093.3
k-Ge simp	294	2219722	1167.7

Trivium (1020 instances)			
Solver	#	Decisions	Time (s)
cryptominisat	713	8344	1.6
Eq* simp	498	-	-
Eq*	494	-	-
k-Ge simp	434	-	-
k-Ge	425	-	-

Figure 7.13. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the seven benchmark families (results for cryptominisat)

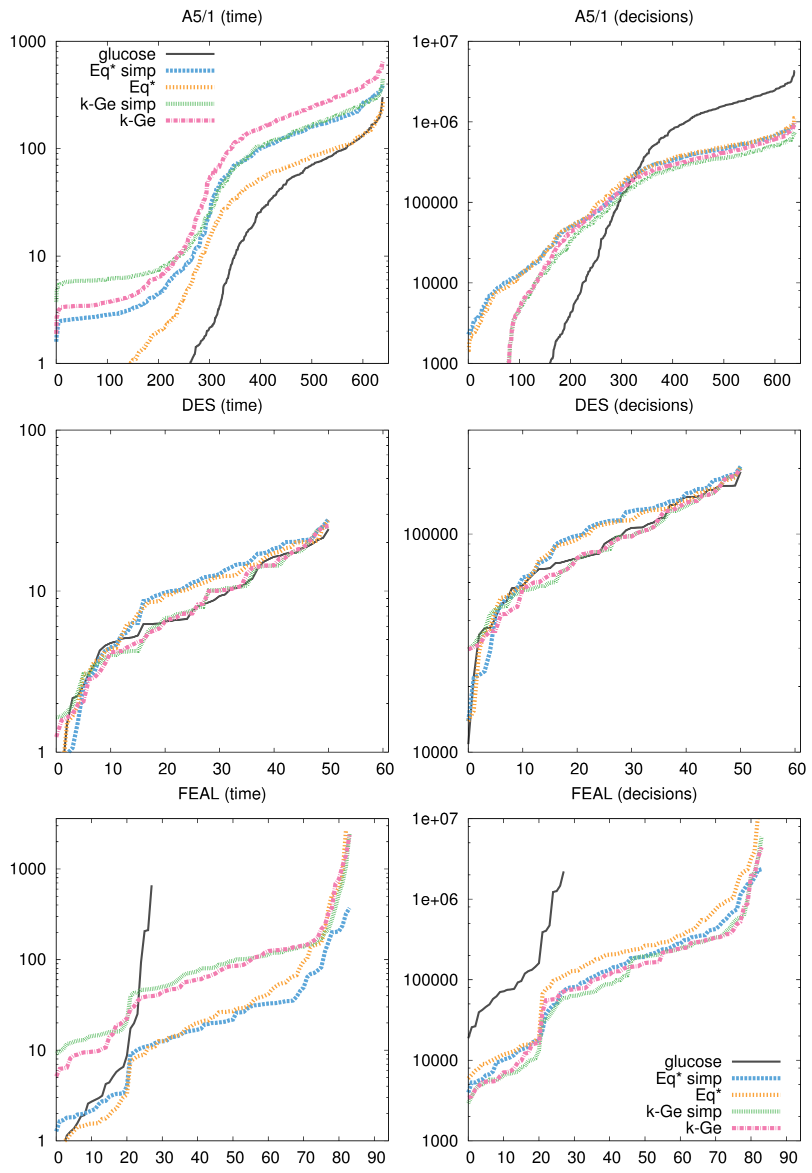


Figure 7.14. Solving time and number of decisions as functions of solved instances (glucose part 1/3)

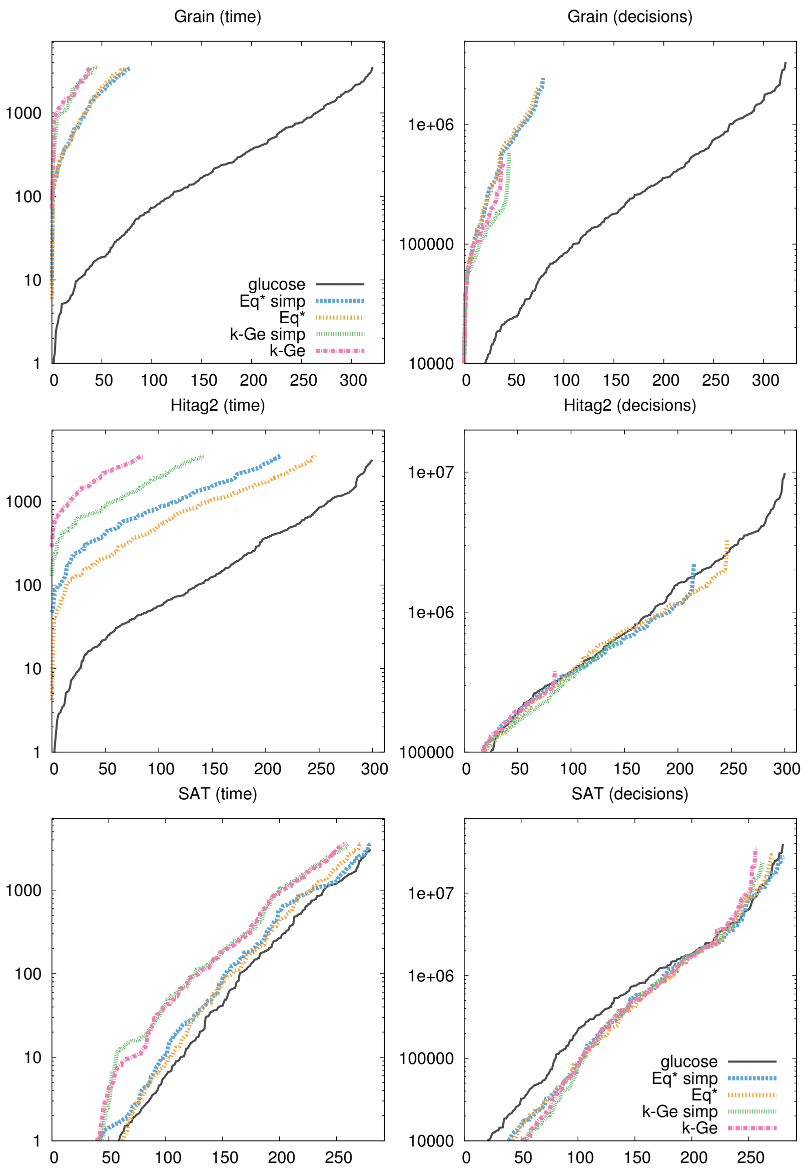


Figure 7.15. Solving time and number of decisions as functions of solved instances (glucose part 2/3)

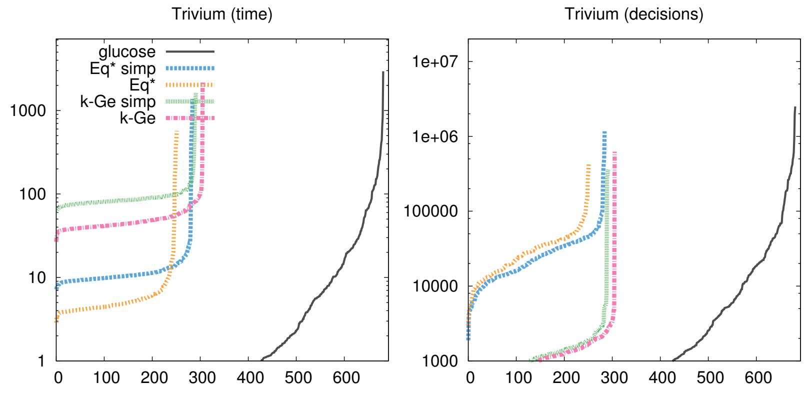


Figure 7.16. Solving time and number of decisions as functions of solved instances (glucose part 3/3)

A5/1 (640 instances)			
Solver	#	Decisions	Time (s)
k-Ge simp	640	148695	43.1
k-Ge	640	183445	69.3
glucose	640	187170	3.4
Eq* simp	640	205392	45.6
Eq*	640	217652	22.8

DES (51 instances)			
Solver	#	Decisions	Time (s)
k-Ge simp	51	84465	7.6
k-Ge	51	85196	7.3
glucose	51	90127	7.5
Eq*	51	110672	10.6
Eq* simp	51	113465	11.3

FEAL (84 instances)			
Solver	#	Decisions	Time (s)
k-Ge simp	84	94335	80.0
k-Ge	84	126654	62.5
Eq* simp	84	143244	18.3
Eq*	83	207108	20.9
glucose	28	-	-

Grain (357 instances)			
Solver	#	Decisions	Time (s)
glucose	323	270533	252.0
Eq* simp	80	-	-
Eq*	76	-	-
k-Ge simp	46	-	-
k-Ge	40	-	-

Hitag2 (306 instances)			
Solver	#	Decisions	Time (s)
glucose	301	726224	127.7
Eq*	247	747539	1068.6
Eq* simp	216	663520	1584.0
k-Ge simp	145	-	-
k-Ge	86	-	-

SAT (474 instances)			
Solver	#	Decisions	Time (s)
Eq* simp	282	3777279	1104.5
glucose	281	4384845	874.1
Eq*	272	4174640	1293.8
k-Ge simp	263	4257933	2166.7
k-Ge	258	5174093	2078.7

Trivium (1020 instances)			
Solver	#	Decisions	Time (s)
glucose	682	3212	2.9
k-Ge	306	-	-
k-Ge simp	292	-	-
Eq* simp	285	-	-
Eq*	252	-	-

Figure 7.17. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the seven benchmark families (results for glucose)

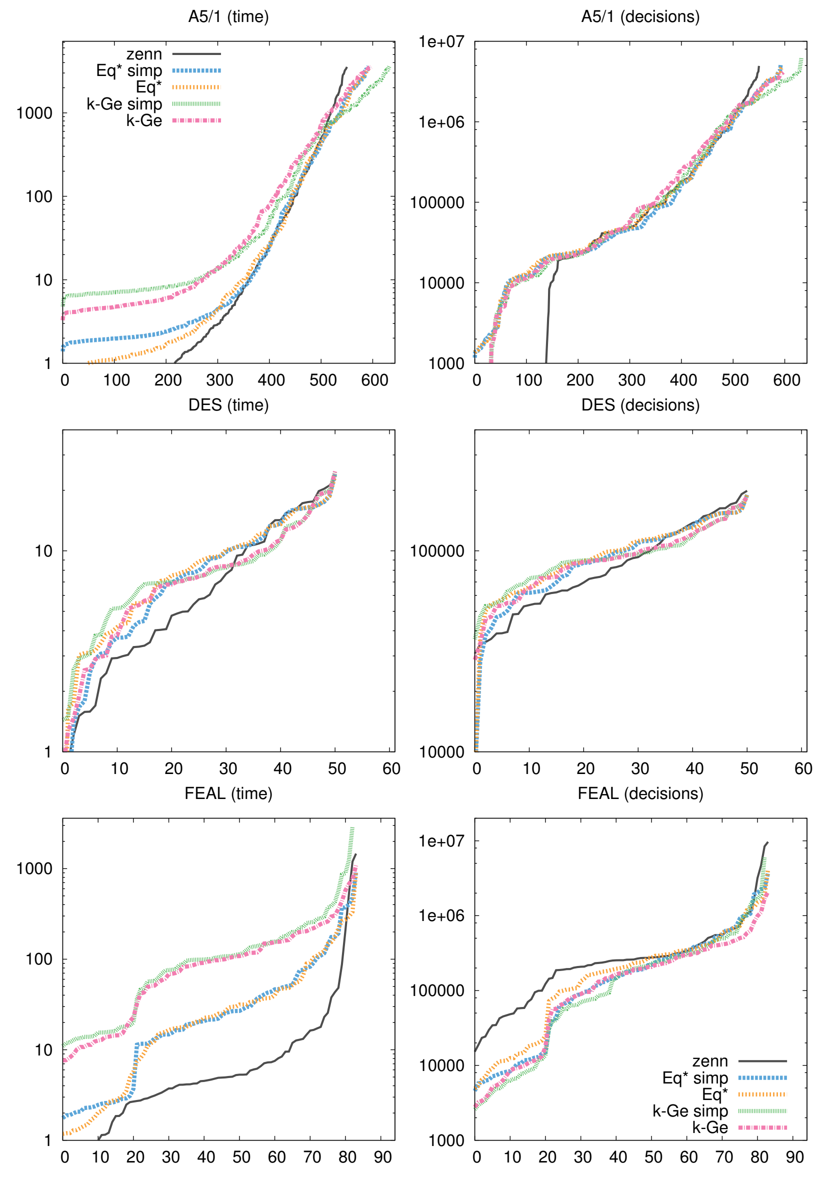


Figure 7.18. Solving time and number of decisions as functions of solved instances (zenn part 1/3)

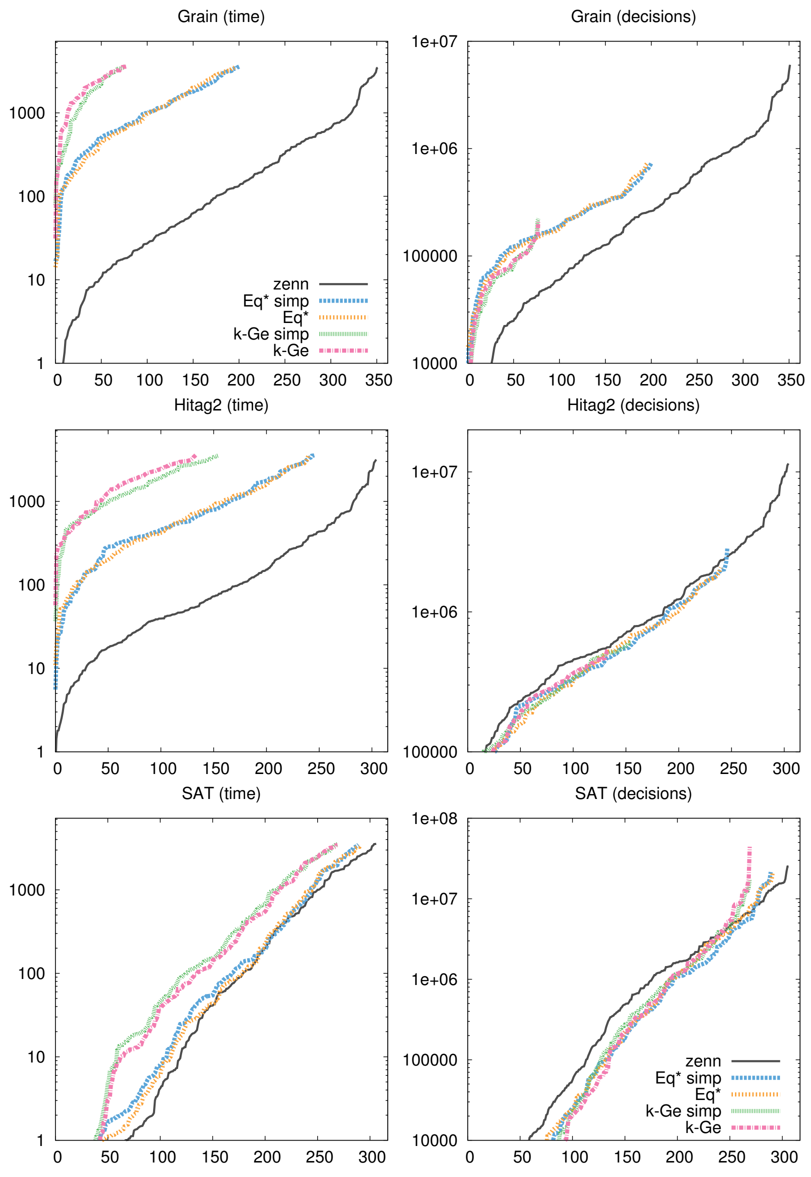


Figure 7.19. Solving time and number of decisions as functions of solved instances (zenn part 2/3)

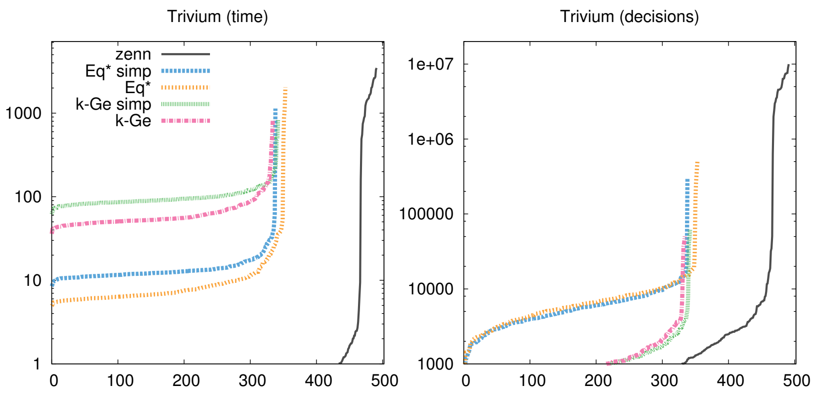


Figure 7.20. Solving time and number of decisions as functions of solved instances (zenn part 3/3)

A5/1 (640 instances)			
Solver	#	Decisions	Time (s)
k-Ge simp	633	71065	16.3
k-Ge	598	84837	17.7
Eq*	595	62646	6.7
Eq* simp	593	48462	5.2
zenn	551	59616	4.0

DES (51 instances)			
Solver	#	Decisions	Time (s)
zenn	51	82050	5.7
k-Ge	51	91097	7.4
k-Ge simp	51	91097	7.5
Eq* simp	51	95106	8.0
Eq*	51	99548	8.8

FEAL (84 instances)			
Solver	#	Decisions	Time (s)
Eq* simp	84	160957	21.5
k-Ge	84	169530	94.5
Eq*	84	201723	22.9
zenn	84	254023	4.6
k-Ge simp	83	167844	100.2

Grain (357 instances)			
Solver	#	Decisions	Time (s)
zenn	352	211980	103.0
Eq* simp	202	444935	2679.7
Eq*	196	501079	2946.7
k-Ge	78	-	-
k-Ge simp	78	-	-

Hitag2 (306 instances)			
Solver	#	Decisions	Time (s)
zenn	305	724854	74.5
Eq* simp	247	553934	799.4
Eq*	241	621766	941.6
k-Ge simp	156	594356	3475.4
k-Ge	134	-	-

SAT (474 instances)			
Solver	#	Decisions	Time (s)
zenn	307	3378537	625.2
Eq*	292	3069530	793.2
Eq* simp	290	2085935	734.5
k-Ge simp	270	3181330	1749.0
k-Ge	270	3184636	1953.3

Trivium (1020 instances)			
Solver	#	Decisions	Time (s)
zenn	492	-	-
Eq*	354	-	-
k-Ge simp	343	-	-
Eq* simp	339	-	-
k-Ge	335	-	-

Figure 7.21. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the seven benchmark families (results for zenn)

A5/1 (640 instances)			
Solver	#	Decisions	Time (s)
Eq* simp	607	24722	9.0
Eq*	605	19724	6.8
k-Ge simp	591	23558	21.4
k-Ge	571	24739	22.3
cryptominisat	504	340275	8.3

DES (51 instances)			
Solver	#	Decisions	Time (s)
k-Ge	51	76706	8.1
k-Ge simp	51	77799	8.0
Eq*	51	92624	11.6
cryptominisat	51	93695	10.9
Eq* simp	51	96072	16.6

FEAL (84 instances)			
Solver	#	Decisions	Time (s)
k-Ge	84	66274	54.8
k-Ge simp	84	68468	76.0
Eq* simp	84	91303	14.9
Eq*	84	95753	14.8
cryptominisat	84	200137	4.4

Grain (357 instances)			
Solver	#	Decisions	Time (s)
cryptominisat	356	253777	46.2
k-Ge simp	54	-	-
k-Ge	41	-	-
Eq*	28	-	-
Eq* simp	23	-	-

Hitag2 (306 instances)			
Solver	#	Decisions	Time (s)
cryptominisat	306	577225	61.9
Eq* simp	257	519855	831.0
Eq*	257	531972	852.6
k-Ge	133	-	-
k-Ge simp	105	-	-

SAT (474 instances)			
Solver	#	Decisions	Time (s)
Eq*	312	2092268	588.6
Eq* simp	309	1961380	642.4
cryptominisat	308	2090916	442.0
k-Ge	300	1754659	1093.3
k-Ge simp	294	2219722	1167.7

Trivium (1020 instances)			
Solver	#	Decisions	Time (s)
cryptominisat	713	8344	1.6
Eq* simp	498	-	-
Eq*	494	-	-
k-Ge simp	434	-	-
k-Ge	425	-	-

Figure 7.22. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the seven benchmark families (results for cryptominisat)

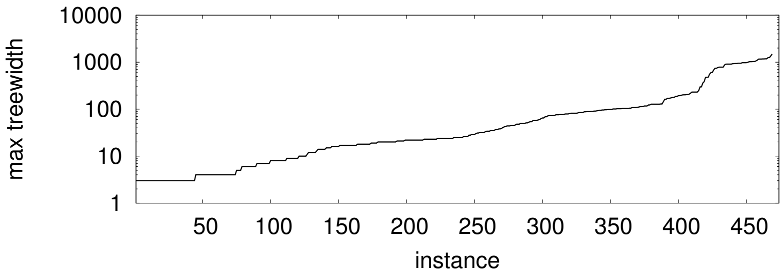


Figure 7.23. Treewidth in SAT 05-11 instances

Solver	#	Decisions	Time(s)
IGJ	640	3099	1.9
glucose	640	187170	3.4
glucose+Eq*	640	217652	22.8
glucose+k-Ge simp	640	148695	43.1
glucose+Eq* simp	640	205392	45.6
glucose+k-Ge	640	183445	69.3
UP+learn	639	20651	4.2
zenn+k-Ge simp	633	71065	16.3
minisat	626	37096	5.0
UP+fcut	617	30036	5.6
UP	613	34910	5.0
UP+pexp	611	32745	5.6
cryptominisat+Eq* simp	607	24722	9.0
cryptominisat+Eq*	605	19724	6.8
SUBST+p	603	5289	7.6
zenn+k-Ge	598	84837	17.7
SUBST	597	3134	5.2
zenn+Eq*	595	62646	6.7
zenn+Eq* simp	593	48462	5.2
cryptominisat+k-Ge simp	591	23558	21.4
cryptominisat+k-Ge	571	24739	22.3
zenn	551	59616	4.0
EC	548	5762	19.4
cryptominisat	504	340275	8.3

Figure 7.24. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the benchmark family A5/1 (640 instances)

Solver	#	Decisions	Time(s)
zenn	51	82050	5.7
glucose+k-Ge	51	85196	7.3
zenn+k-Ge	51	91097	7.4
zenn+k-Ge simp	51	91097	7.5
glucose	51	90127	7.5
glucose+k-Ge simp	51	84465	7.6
zenn+Eq* simp	51	95106	8.0
cryptominisat+k-Ge simp	51	77799	8.0
cryptominisat+k-Ge	51	76706	8.1
zenn+Eq*	51	99548	8.8
glucose+Eq*	51	110672	10.6
cryptominisat	51	93695	10.9
glucose+Eq* simp	51	113465	11.3
cryptominisat+Eq*	51	92624	11.6
UP	51	72062	12.2
minisat	51	97652	14.5
UP+learn	51	90122	16.2
UP+fcut	51	95881	16.6
cryptominisat+Eq* simp	51	96072	16.6
SUBST+p	51	90439	17.0
SUBST	51	91826	17.3
UP+pexp	51	91832	17.6
IGJ	51	89991	19.2
EC	51	249363	73.7

Figure 7.25. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the benchmark family DES (51 instances)

Solver	#	Decisions	Time(s)
cryptominisat	84	200137	4.4
zenn	84	254023	4.6
cryptominisat+Eq*	84	95753	14.8
cryptominisat+Eq* simp	84	91303	14.9
glucose+Eq* simp	84	143244	18.3
zenn+Eq* simp	84	160957	21.5
zenn+Eq*	84	201723	22.9
SUBST+p	84	200141	25.6
minisat	84	811828	54.1
cryptominisat+k-Ge	84	66274	54.8
glucose+k-Ge	84	126654	62.5
cryptominisat+k-Ge simp	84	68468	76.0
glucose+k-Ge simp	84	94335	80.0
zenn+k-Ge	84	169530	94.5
UP	84	1594754	145.7
UP+fcut	84	1426547	150.8
glucose+Eq*	83	207108	20.9
zenn+k-Ge simp	83	167844	100.2
UP+pexp	83	1131949	109.2
IGJ	79	73113	92.8
UP+learn	35	-	-
glucose	28	-	-
SUBST	21	-	-
EC	21	-	-

Figure 7.26. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the benchmark family FEAL (84 instances)

Solver	#	Decisions	Time(s)
cryptominisat	356	253777	46.2
zenn	352	211980	103.0
glucose	323	270533	252.0
minisat	323	265328	282.1
UP+pexp	315	271660	320.8
UP	313	261088	275.1
UP+fcut	306	244842	267.9
UP+learn	290	153123	420.0
SUBST	222	127654	1759.7
zenn+Eq* simp	202	444935	2679.7
SUBST+p	198	247511	2773.1
zenn+Eq*	196	501079	2946.7
IGJ	186	239424	3267.4
EC	165	-	-
glucose+Eq* simp	79	-	-
zenn+k-Ge	78	-	-
zenn+k-Ge simp	77	-	-
glucose+Eq*	76	-	-
cryptominisat+k-Ge simp	54	-	-
glucose+k-Ge simp	46	-	-
cryptominisat+k-Ge	41	-	-
glucose+k-Ge	40	-	-
cryptominisat+Eq*	28	-	-
cryptominisat+Eq* simp	23	-	-

Figure 7.27. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the benchmark family Grain (357 instances)

Solver	#	Decisions	Time(s)
cryptominisat	306	577225	61.9
zenn	305	724854	74.5
glucose	301	726224	127.7
minisat	295	1100849	280.0
UP+pexp	284	1358081	459.0
UP+learn	282	1028143	357.2
UP	282	1404806	488.1
UP+fcut	281	1247305	434.0
cryptominisat+Eq* simp	257	519855	831.0
cryptominisat+Eq*	257	531972	852.6
zenn+Eq* simp	247	553934	799.4
SUBST+p	247	1661239	1068.4
glucose+Eq*	247	747539	1068.6
zenn+Eq*	241	621766	941.6
glucose+Eq* simp	215	663520	1584.0
SUBST	190	2631385	2219.3
IGJ	171	1556543	3098.2
zenn+k-Ge simp	155	600458	3475.4
glucose+k-Ge simp	143	-	-
zenn+k-Ge	134	-	-
cryptominisat+k-Ge	133	-	-
cryptominisat+k-Ge simp	104	-	-
EC	102	-	-
glucose+k-Ge	86	-	-

Figure 7.28. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the benchmark family Hitag2 (306 instances)

Solver	#	Decisions	Time(s)
cryptominisat+Eq*	312	2092268	588.6
cryptominisat+Eq* simp	308	1972539	642.4
cryptominisat	307	2127930	442.0
zenn	307	3378537	625.2
UP+learn	301	1930731	428.3
cryptominisat+k-Ge	300	1754659	1093.3
cryptominisat+k-Ge simp	294	2219722	1167.7
zenn+Eq*	292	3069530	793.2
IGJ	292	1397013	1069.5
zenn+Eq* simp	290	2085935	734.5
glucose+Eq* simp	282	3777279	1104.5
glucose	281	4384845	874.1
minisat	279	14857305	1401.7
glucose+Eq*	272	4174640	1293.8
UP	271	18892454	1613.3
zenn+k-Ge simp	270	3181330	1749.0
zenn+k-Ge	270	3184636	1953.3
UP+fcut	269	22568178	2231.6
UP+pexp	267	28097432	2007.9
SUBST+p	266	14369875	2207.2
glucose+k-Ge simp	261	5121320	2166.7
glucose+k-Ge	258	5174093	2078.7
SUBST	211	-	-
EC	130	-	-

Figure 7.29. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the benchmark family SAT (474 instances)

Solver	#	Decisions	Time(s)
UP+learn	907	6739	8.9
minisat	902	9361	3.8
UP	893	10751	5.8
UP+fcut	881	9718	5.2
UP+pexp	880	9442	5.1
IGJ	873	2569	21.3
SUBST	826	2872	47.5
SUBST+p	803	2985	60.4
EC	735	3732	142.0
cryptominisat	713	8344	1.6
glucose	682	3212	2.9
cryptominisat+Eq* simp	498	-	-
cryptominisat+Eq*	494	-	-
zenn	492	-	-
cryptominisat+k-Ge simp	434	-	-
cryptominisat+k-Ge	425	-	-
zenn+Eq*	354	-	-
zenn+k-Ge simp	343	-	-
zenn+Eq* simp	339	-	-
zenn+k-Ge	335	-	-
glucose+k-Ge	306	-	-
glucose+k-Ge simp	292	-	-
glucose+Eq* simp	285	-	-
glucose+Eq*	252	-	-

Figure 7.30. Number of solved instances (#), median decisions, and median solving time (timeout 1h) on the benchmark family Trivium (1020 instances)

8. Conclusions

The research problem studied in this thesis is to develop methods to solve a relevant class of the propositional satisfiability (SAT) problem: to assign the variables of a propositional formula consisting of clauses and parity (xor) constraints in a way that the formula evaluates to ”true“ or to conclude that it is not possible. Such problems, common in application domains such as circuit verification, bounded model checking, logical cryptanalysis, and approximate model counting, can be challenging for modern conflict-driven clause learning SAT solvers without specialized parity reasoning techniques.

The research problem is addressed by basing on the previously introduced DPLL(XOR) framework that allows the results in this thesis to be used with small effort in existing and future conflict-driven clause-learning SAT solvers. The thesis develops three new xor-deduction systems, UP, EC and IGJ, and a number of other techniques to enhance them and relate them to each other and to other systems, such as the previously introduced xor-deduction system Subst. The new xor-deduction systems differ in proof system strength, implementation efficiency and how implying clauses needed for conflict analysis in SAT solver are computed.

The baseline xor-deduction system UP that performs plain unit propagation on xor-constraints offers an efficiently implementable reference solution technique to which other xor-deduction systems and additional methods can be compared. The xor-deduction system EC performs a form of equivalence reasoning equally strong to Subst that allows to compute shorter xor-explanations than Subst. The xor-deduction system IGJ performs incremental Gauss-Jordan elimination, a complete parity reasoning technique.

The structure of the problem has a strong impact on which xor-deduction system performs the best. It is found that problems consisting mostly of complex tightly connected xor-constraints are solved the fastest by Gauss-Jordan elimination, while problems exhibiting easily recognizable “tree-like” structure do not benefit from proof systems stronger than unit propagation, and are thus UP-deducible. Naturally, there are xor-constraint conjunctions for which equivalence reasoning is a complete parity reasoning technique. Some of such Subst-deducible (or EC-deducible) xor-constraint conjunctions can be detected by our fast structural test based on the close connection between equivalence reasoning and cycles in the (xor) constraint graph. It remains open whether more accurate fast tests for UP-deducibility and Subst-deducibility can be developed, or it is computationally beyond practical applicability, e.g. coNP-complete. One could also envision incomplete xor-deduction systems stronger than Subst and develop corresponding fast structural tests to always use the optimal xor-deduction system for given a problem to solve.

Besides proof system strength, the quality of implying clauses given by an xor-deduction system affects the overall performance of SAT solving. The thesis develops new techniques for analyzing xor-derivations produced by the xor-deduction systems UP, Subst, and EC allowing one to obtain parity explanations for xor-implied literals. Parity explanations translate to shorter implying clauses, give very short unsatisfiability proofs for formulas hard for resolution, and can also used to derive and learn new xor-constraints. Provided that additional non-deterministic assumptions can be made, it is shown that parity explanations on plain unit propagation xor-derivations can simulate a complete Gauss-Jordan elimination parity reasoning engine on a restricted but practically relevant class of xor-constraint conjunctions. It is left for future work to discover how parity explanations be exploited to greater degree and to find out whether parity explanations on plain unit propagation derivations in fact simulate Gauss-Jordan elimination on all xor-constraint conjunctions.

Even if it is possible to detect accurately which xor-deduction system to use with a given xor-constraint conjunction, it may be of limited use if only a small part of the problem requires stronger parity reasoning. The thesis develops methods to decompose xor-constraint conjunctions into separate subproblems that can be handled separately to some extent. Primarily motivated by reducing the memory usage of incremental Gauss-Jordan elimination when using dense matrix representation, the new methods to decompose xor-constraint conjunctions are also useful in conjunction with the approximating structural tests to detect UP-deducibility and Subst-deducibility to allow more fine-grained selection of solving technique for different parts of the problem.

To leverage SAT solvers without parity reasoning techniques, the thesis develops four translations that enable unit propagation to simulate stronger xor-deduction systems by adding redundant xor-constraints in the problem description. It is shown that without auxiliary variables, an exponential number of additional xor-constraints are needed to simulate equivalence reasoning while with auxiliary variables, a cubic number of additional xor-constraints suffices. This number may often be reduced significantly by using our translations that take into account the structure of the xor-constraint conjunction, and by our simplification technique that preserves literals implied by unit propagation. While it is technically possible to simulate complete parity reasoning engine, e.g. Gauss-Jordan elimination, by adding redundant xor-constraints, the resulting formula may be prohibitively large. However, it is shown that complete parity reasoning engine can be simulated with a polynomial number of additional xor-constraints for instances of bounded treewidth. In practice, a viable solution technique may be to use these translations partially to strengthen unit propagation only when it can be done with a limited number of xor-constraints.

The practical and theoretical results in this thesis may be applied directly to solve even larger problems in the application domains where parity constraints are already used as a part of the modeling language. The theoretical results provide a foundation for building next generation SAT solvers capable of handling xor-constraints effectively. Finally, the results encourage using parity constraints as a part of the modeling language despite their reputation for hindering solving performance, because parity constraints provide structure which can be exploited.

Bibliography

S. B. Akers. Binary decision diagrams. *IEEE Transactions on Compututers*, 27(6):509–516, June 1978.

Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, 2002.

Fahiem Bacchus. Enhancing Davis Putnam with extended binary clause reasoning. In *Proceedings of the 18th AAAI Conference on Artificial Intelligence (AAAI-2002)*, pages 613–619. AAAI Press, 2002.

Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Biere et al. Biere *et al.* [2009], pages 825–885.

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, *Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2000.

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

Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In Bimal K. Roy and Eiji Okamoto, editors, *INDOCRYPT*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer, 2000.

Lucas Bordeaux and João Marques-Silva. Knowledge compilation with empowerment. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 612–624. Springer Berlin Heidelberg, 2012.

Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi Junttila, Silvio Ranise, Peter van Rossum, and Roberto Sebastiani. Efficient theory combination via boolean search. *Information and Computation*, 204(10):1493 – 1525, 2006.

Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

J. Chen. Building a hybrid SAT solver via conflict-driven, look-ahead and XOR reasoning techniques. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 298–311. Springer, 2009.

Nicolas Courtois, Sean O'Neil, and Jean-Jacques Quisquater. Practical algebraic attacks on the hitag2 stream cipher. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio A. Ardagna, editors, *Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 167–176. Springer, 2009.

William Craig. Linear reasoning. a new form of the herbrand-gentzen theorem. *Journal of Symbolic Logic*, 22(3):250–268, 09 1957.

David Cyrluk, M. Oliver Möller, and Harald Rueß. An efficient decision procedure for the theory of fixed-sized bit-vectors. In Orna Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 1997.

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

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

Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

DES. *Data encryption standard*. U. S. Department of Commerce, National Bureau of Standards, 1977.

David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM*, 52(3):365–473, 2005.

Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for DPLL(T). In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

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

Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure, and Lakhdar Sais. Reducing hard SAT instances to polynomial ones. In *IEEE International Conference on Information Reuse and Integration, IRI 2007*, pages 18–23. IEEE Systems, Man, and Cybernetics Society, 2007.

Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of the 8th National conference on Artificial intelligence - Volume 1*, pages 4–9. AAAI Press, 1990.

Eugene Goldberg and Yakov Novikov. Berkmin: A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12):1549–1561, 2007.

Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In Jack Mostow and Chuck Rich, editors, *Proceedings of the 15th National / 10th Conference on Artificial Intelligence / Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 431–437. AAAI Press / The MIT Press, 1998.

Matthew Gwynne and Oliver Kullmann. On SAT representations of XOR constraints. *CoRR*, abs/1309.3060, 2013.

Matthew Gwynne and Oliver Kullmann. On sat representations of xor constraints. In Adrian-Horia Dediu, Carlos Martín-Vide, José-Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications*, volume 8370 of *Lecture Notes in Computer Science*, pages 409–420. Springer International Publishing, 2014.

Cheng-Shen Han and Jie-Hong Roland Jiang. When boolean satisfiability meets gaussian elimination in a simplex way. In *Proceedings of the 24th International Conference on Computer Aided Verification*, CAV'12, pages 410–426, Berlin, Heidelberg, 2012. Springer-Verlag.

Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, 2007.

Marijn Heule and Hans van Maaren. Aligning CNF- and equivalence-reasoning. In Hoos and Mitchell Hoos and Mitchell [2005], pages 145–156.

Marijn Heule, Mark Dufour, Joris van Zwieten, and Hans van Maaren. March_eq: Implementing additional reasoning into an efficient look-ahead SAT solver. In Hoos and Mitchell Hoos and Mitchell [2005], pages 345–359.

Holger H. Hoos and David G. Mitchell, editors. *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*. Springer, 2005.

John E. Hopcroft and Robert Endre Tarjan. Efficient algorithms for graph manipulation [h] (algorithm 447). *Communications of the ACM*, 16(6):372–378, 1973.

George Katsirelos and Laurent Simon. Learning polynomials over gf(2) in a sat solver. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, SAT'12, pages 496–497, Berlin, Heidelberg, 2012. Springer-Verlag.

Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Extending clause learning DPLL with parity reasoning. In *Proceedings of the 19th European Conference on Artificial Intelligence, ECAI 2010*, pages 21–26. IOS Press, 2010.

Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Extending clause learning SAT solvers with complete parity reasoning (extended version). arXiv document arXiv:1207.0988 [cs.LO], 2012.

Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Simulating parity reasoning (extended version). arXiv document arXiv:1311.4289 [cs.LO], 2013.

Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Classifying and propagating parity constraints (extended version). arXiv document arXiv:1406.4698 [cs.LO], 2014.

Tero Laitinen, Tommi Junttila, and Ilkka Niemelä. Conflict-driven XOR-clause learning (extended version). arXiv document arXiv:1407.6571 [cs.LO], 2014.

Matthew D. T. Lewis, Tobias Schubert, and Bernd Becker. Speedup techniques utilized in modern SAT solvers. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 437–443. Springer, 2005.

Chu Min Li. Equivalency reasoning to solve a class of hard SAT problems. *Information Processing Letters*, 76(1-2):75–81, 2000.

Chu Min Li. Integrating equivalency reasoning into Davis-Putnam procedure. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 291–296. AAAI Press / The MIT Press, 2000.

Chu Min Li. Equivalent literal propagation in the DLL procedure. *Discrete Applied Mathematics*, 130(2):251–276, 2003.

Vasco M. Manquinho and Olivier Roussel. The first evaluation of pseudo-boolean solvers (PB'05). *Journal on Satisfiability, Boolean Modeling and Computation*, 2:103–143, 2006.

George H. Mealy. A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

Shoji Miyaguchi. The feal cipher family. In AlfredJ. Menezes and ScottA. Vanstone, editors, *Advances in Cryptology-CRYPT0' 90*, volume 537 of *Lecture Notes in Computer Science*, pages 628–638. Springer, 1991.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001.

Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. *Information and Computing*, 205(4):557–580, 2007.

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.

Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Sais. Recovering and exploiting structural knowledge from CNF formulas. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2002.

Judea Pearl. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the National Conference on Artificial Intelligence, AAAI 1982*, pages 133–136. AAAI Press, 1982.

Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, 2011.

Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, Burnaby, BC, Canada, 2004.

Marko Samer and Stefan Szeider. Fixed-parameter tractability. In Biere et al. Biere *et al.* [2009], pages 425–454.

Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *Journal of Computer and System Sciences*, 76(2):103 – 114, 2010.

Roberto Sebastiani. Lazy satisability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(3-4):141–224, 2007.

João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *Proceedings of the 1996 International Conference on Computer-Aided Design, November 10-14, 1996, San Jose, CA, USA*, pages 220–227. ACM and IEEE Computer Society, 1996.

João P. Marques Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In Biere et al. Biere *et al.* [2009], pages 131–153.

M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

Mate Soos. Enhanced gaussian elimination in DPLL-based SAT solvers. In *Pragmatics of SAT*, pages 1–1, Edinburgh, Scotland, GB, July 2010.

G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125, 1968.

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

Sean Andrew Weaver. *Satisfiability advancements enabled by state machines*. PhD thesis, University of Cincinnati, Cincinnati, OH, USA, 2012. AAI3554401.

Lintao Zhang and Sharad Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*, pages 880–885. IEEE, 2003.

Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in boolean satisfiability solver. In *Proceedings of the 2001 International Conference on Computer-Aided Design, November 4-8, 2001, San Jose, CA, USA*, pages 279–285. ACM, 2001.

Aalto-DD 177/2014

ISBN 978-952-60-5944-0 (printed)
ISBN 978-952-60-5945-7 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Information and Computer Science
www.aalto.fi