Department of Computer Science
Series of Publications A
Report A-2019-12

Implicit Hitting Set Algorithms for Constraint Optimization

Paul Saikko

Doctoral dissertation, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki, in Auditorium B123, Exactum, on December 2nd, 2019, at 12 o'clock noon.

University of Helsinki
Finland

Supervisors

Associate Professor Matti Järvisalo, University of Helsinki, Finland
Professor Petri Myllymäki, University of Helsinki, Finland

Pre-examiners

Professor Martin Gebser, University of Klagenfurt, Austria
Professor David Mitchell, Simon Fraser University, Canada

Opponent

Professor Laurent Simon, University of Bordeaux, France

Custos

Associate Professor Matti Järvisalo, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: https://www.helsinki.fi/en/computer-science
Telephone: +358 2941 911

ISSN 1238-8645
ISBN 978-951-51-5668-6 (paperback)
ISBN 978-951-51-5669-3 (PDF)
Helsinki 2019
Unigrafia

Implicit Hitting Set Algorithms for Constraint Optimization

Paul Saikko

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
paul.saikko@helsinki.fi
https://www.cs.helsinki.fi/u/psaikko/

PhD Thesis, Series of Publications A, Report A-2019-12
Helsinki, November 2019, 70+54 pages
ISSN 1238-8645
ISBN 978-951-51-5668-6 (paperback)
ISBN 978-951-51-5669-3 (PDF)

Abstract

Computationally hard optimization problems are commonplace not only in theory but also in practice in many real-world domains. Even determining whether a solution exists can be NP-complete or harder. Good, ideally globally optimal, solutions to instances of such problems can save money, time, or other resources.

We focus on a particular generic framework for solving constraint optimization problems, the so-called implicit hitting set (IHS) approach. The approach is based on a theory of duality between solutions and sets of mutually conflicting constraints underlying a problem. Recent years have seen a number of new instantiations of the IHS approach for various problems and constraint languages.

As the main contributions, we present novel instantiations of this generic algorithmic approach to four different NP-hard problem domains: maximum satisfiability (MaxSAT), learning optimal causal graphs, propositional abduction, and answer set programming (ASP). For MaxSAT, we build on an existing IHS algorithm with a fresh implementation and new methods for integrating preprocessing. We study a specific application of this IHS approach to MaxSAT for learning optimal causal graphs. In particular we develop a number of domain-specific search techniques to specialize the IHS algorithm for the problem. Furthermore, we consider two optimization settings where the corresponding decision problem is beyond NP, in these

cases Σ_2^P-hard. In the first, we compute optimal explanations for propositional abduction problems. In the second, we solve optimization problems expressed as answer set programs with disjunctive rules.

For each problem domain, we empirically evaluate the resulting algorithm and contribute an open-source implementation. These implementations improve or complement the state of the art in their respective domains.

Computing Reviews (2012) Categories and Subject Descriptors:
Mathematics of computing→Combinatorial optimization
Theory of computation→Constraint and logic programming
Artificial intelligence→Logic programming and answer set programming

General Terms:
algorithms, exact optimization, declarative programming

Additional Key Words and Phrases:
constraint optimization, implicit hitting set algorithms, satisfiability, maximum satisfiability, MaxSAT, propositional logic, abduction, causal discovery

Acknowledgements

The foundations for this PhD work started when I joined the Constraint Reasoning and Optimization (CoReO) group as a research assistant over four years ago. I thank the members of the CoReO group for providing a productive research environment and interesting algorithmic problems. I am most grateful to my advisors, Matti Järvisalo for generously sharing his time and expertise with me since the beginning, and Petri Myllymäki for the many roles he has played in making this work possible.

This dissertation is the result of collaborations with my co-authors Matti Järvisalo, Jeremias Berg, Johannes Wallner, Antti Hyttinen, Carmine Dodaro, and Mario Alviano. I am privileged to have published work with them, as well as Brandon Malone, Fahiem Bacchus, and Tuukka Korhonen during my doctoral studies.

I thank my pre-examiners Martin Gebser and David Mitchell for taking the time to review this work, and Laurent Simon for agreeing to be the opponent in the public examination.

This work would likely never have been completed without the financial stability and support provided by the Doctoral Programme in Computer Science (DoCS). I thank the board of the programme for selecting me for a salaried doctoral candidate position, and research coordinator Pirjo Moen for her feedback in finalizing this thesis. I am also grateful for the support of the Finnish Foundation for Technology Promotion (TES).

I would like to extend my thanks to TKO-äly ry and the student room community for the many distractions from my research work. In particular Topi Talvitie, whose parallel PhD journey has been a source of motivation.

Finally, I thank my parents Timo and Stina for their love and support, and for giving me the opportunity to pursue my interests.

Helsinki, November 2019

Paul Saikko

Contents

Chapter 1

Introduction

In this thesis we develop constraint optimization procedures for computationally difficult problems. Optimization problems are found where many possible solutions exist but we prefer some of them over others. This preference might be for e.g. the cheapest, fastest, or most probable solution. Many interesting cases are computationally difficult, even determining if a solution exists can be NP-complete or in a complexity class beyond NP.

Various types of approaches for solving optimization problems exist. Local search algorithms [81] repeatedly explore the neighborhood of a current best solution for better results. They are computationally efficient and scalable, but do not give any guarantees on the quality of a found solution. Approximation algorithms [155] give stronger guarantees on how much the cost of a solution differs from an optimal one. Exact optimization algorithms find a provably *globally optimal* solution but may not terminate within a "reasonable" amount of time on all inputs. Naturally we would like to obtain globally optimal solutions where it is possible to compute them. Compared to an approximation, an optimal solution can save time, money, or other resources depending on the application. In cases where it is not feasible to compute an optimal solution to the original problem, a simplified version can be solved exactly to get a better approximation.

As an alternative to application-specific algorithms, hard optimization problems can be solved *declaratively*. Specifically we consider different declarative constraint programming paradigms, where problems are described in a generic manner as a set of constraints. The set of constraints is then solved by an algorithm, or solver, for that constraint language. The types of constraints vary by domain. Examples include linear inequalities as linear programming (LP) [142] constraints and disjunctions of literals as Boolean satisfiability (SAT) [63] constraints. Constraint programming approaches are popular for a number of reasons. The task of modeling a

problem as a set of constraints can be more straightforward than devising an algorithm. Due to the large body of work behind them, solvers targeting these constraint languages may even be more efficient than an algorithm specialized for the problem. Sets of constraints are very extensible, as new constraints can be added without changes to the solver.

Integer programming (IP) [142] has long been a standard declarative approach for solving hard optimization problems. Integer programs consist of a linear objective function and linear inequalities over integer (and continuous) variables. Logic-based optimization languages such as maximum satisfiability (MaxSAT) [101] have recently seen increased use following the success of practical algorithms (solvers) for SAT. Solvers for optimization with logic-based languages are commonly implemented iteratively as a sequence of invocations of a decision procedure.

In this thesis we focus on implicit hitting set (IHS) algorithms [34, 113], which combine IP and logic-based procedures by separating the decision and optimization aspects of a problem. The approach is intuitively attractive because e.g. SAT solvers excel at proving unsatisfiability (non-existence of a solution) while IP solvers are well-suited to dealing with objective functions with large coefficients. An implicit hitting set procedure iteratively identifies subsets of constraints that cannot be simultaneously satisfied (a decision problem) and computes the least costly way to leave out a constraint from each of those subsets (an optimization problem).

The fundamental ideas behind implicit hitting set algorithms can be traced back at least to the works of Reggia et al. [131] and Reiter [132] on diagnosis. Reggia et al. proposed a set-covering based diagnostic system motivated by applications in medicine. Reiter's work on fault diagnosis of systems (e.g. digital circuits) described in first-order logic aimed to enumerate explanations of observed abnormal operation of a system. Recently this work on diagnosis has been revisited in an implicit hitting set context with an exploration of the hitting set duality between conflicts and diagnoses [147] and a new approach for diagnosing multiple observations [86].

Parker and Ryan [121] proposed an implicit hitting set algorithm for solving the maximum-weight feasible subsystem (MAX-FS) problem for an inconsistent set of linear inequalities $A\mathbf{x} \leq \mathbf{b}$. Their approach solves the problem by finding a minimum-weight cover (i.e. hitting set) of irreducible inconsistent subsystems (IIS) [35, 154]. More recent work by Chandrasekaran et al. [34] and Moreno-Centeno and Karp [113] proposed implicit hitting set algorithms as a general framework, applicable to a wide range of problems including Max Cut, Max-2SAT, and the traveling salesperson problem. In contrast to this thesis, both restrict their approach to

polynomial-time oracles for testing the feasibility of a hitting set, emphasizing the difficulty of computing minimum hitting sets instead.

Implicit hitting set algorithms have seen practical use in a variety of problem domains. In particular, instantiations of the implicit hitting set framework have been developed for many constraint programming paradigms, including MaxSAT [44], Horn MaxSAT [105], maximum satisfiability modulo theories (MaxSMT) [37, 62], quantified MaxSAT (QMaxSAT) [85], and weighted constraint satisfaction problems (WCSP) [48]. An application of an implicit hitting set algorithm can also bear resemblance [45, 48] to a logic-based Benders decomposition [80]. Another notable application is in computing smallest minimal unsatisfiable subsets (SMUS) [88].

The more general concept of hitting set duality is also widely explored in the context of MUS computation and enumeration [13,18,102,129]. Likewise algorithms exploiting this type of duality have been proposed for enumeration in a data mining setting [72, 73, 153]. A similar hitting set duality between prime implicates and prime implicants [133] has been used in compilation of non-clausal formulas [128] and computing Horn least upper bounds [112].

1.1 Focus of the Thesis

We investigate the use of implicit hitting set algorithms for optimization problems in NP and beyond. A special focus is placed on practical considerations and heuristics for implementing IHS algorithms. We contribute an open-source implementation for each use of the IHS algorithm studied in this thesis.

This thesis is based on the following original publications, which we will refer to as Papers I-V. The papers are reprinted at the end of the print version of the thesis.

I Paul Saikko, Jeremias Berg, and Matti Järvisalo, **LMHS: A SAT-IP Hybrid MaxSAT Solver**. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing*, volume 9710 of Lecture Notes in Computer Science, pages 539–546. Springer, 2016.

II Jeremias Berg, Paul Saikko, and Matti Järvisalo, **Improving the Effectiveness of SAT-Based Preprocessing for MaxSAT**. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 239–245. AAAI Press, 2015.

III Antti Hyttinen, Paul Saikko, and Matti Järvisalo, **A Core-Guided Approach to Learning Optimal Causal Graphs**. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 645–651. IJCAI, 2017.

IV Paul Saikko, Johannes P. Wallner, and Matti Järvisalo, **Implicit Hitting Set Algorithms for Reasoning Beyond NP**. In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 104–113. AAAI Press, 2016.

V Paul Saikko, Carmine Dodaro, Mario Alviano, and Matti Järvisalo, **A Hybrid Approach to Optimization in Answer Set Programming**. In *Proceedings of the Sixteenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 32–41. AAAI Press, 2018.

In particular, we consider applications of the IHS approach to MaxSAT (Papers I and II), causal structure discovery (Paper III), propositional abduction (Paper IV), and answer set programming (Paper V).

Papers I and II build on earlier work on implicit hitting set algorithms [43–46] and preprocessing for MaxSAT [23,24]. Paper I describes the LMHS MaxSAT solver and Paper II details the integration of preprocessing with the implicit hitting set algorithm. Our work takes better advantage of preprocessing by instantiating the implicit hitting set algorithm to optimize directly over label variables introduced by preprocessing. This tight integration of preprocessing contributed to the success of LMHS in the 2015 MaxSAT evaluation.

In Paper III we present a domain-specific application of the implicit hitting set algorithm. We instantiate the algorithm for the task of computing optimal causal graphs, showing that domain knowledge can be integrated with the implicit hitting set algorithm to enhance it. Our approach benefits from both established SAT and IP solver technology as well as search techniques specific to causal graphs. Domain-specific cores, an incremental encoding, and bounds-based hardening all contribute to the performance of the resulting solver, which surpassed the state of the art for the problem.

Paper IV generalizes the implicit hitting set framework [34, 113] to optimization problems beyond NP. We demonstrate the viability of the approach by instantiating the framework for propositional abduction, which is a Σ_2^P-hard problem [56]. For empirical evidence of its effectiveness, we compare our algorithm to an approach based on a disjunctive answer set programming encoding.

Paper V instantiates the implicit hitting set approach for optimization in answer set programming. We take advantage of recent core-based algorithms to realize a practical implementation of the IHS algorithm. With support for disjunctive rules, this also applies the implicit hitting set approach to a domain beyond NP. In addition to previously used search techniques, we develop improvements specific to answer set programming based on the Clark's completion of a program. The resulting solver improves upon existing core-based solvers for ASP.

Beyond the articles included in this thesis, the author has made further contributions as part of the doctoral studies. These include further work on preprocessing for MaxSAT [28, 29, 97] which does not focus on the implicit hitting set approach, work on reduced-cost fixing for MaxSAT [16, 17] and an application of the LMHS solver for computing optimal cutting planes [137].

1.2 Author Contributions

All papers were jointly written by their authors. Other contributions by the present author are as follows:

Paper I: The present author implemented and maintained the LMHS solver, submitted it in the MaxSAT evaluations, and ran all of the experiments.

Paper II: The present author integrated the preprocessor into the LMHS solver. The use of the preprocessing techniques in a MaxSAT context was investigated by Jeremias Berg.

Paper III: The Dseptor system was implemented jointly with Antti Hyttinen as an extension of the LMHS solver of the present author.

Paper IV: The present author implemented the AbHS procedure, developed the improvements of AbHS+, and ran all of the experiments. The computational complexity results are due to Johannes Wallner.

Paper V: The present author implemented the ASP-HS solver, using the WASP solver of Carmine Dodaro and Mario Alviano, and ran all of the experiments. In-depth analysis of runtime results was done jointly with Matti Järvisalo.

1.3 Organization of the Thesis

We begin with a short review of concepts useful for the main contributions of the thesis. Chapter 2 consists of preliminaries on satisfiability, integer programming, and hitting sets. Chapter 3 then introduces the implicit hitting set problem and the generic implicit hitting set approach, which we instantiate in Papers I–V. The main contributions of the thesis are presented in Chapters 4–7. Chapters 4 and 5 consider the NP-hard problems of Papers I–III, while Chapters 6 and 7 look at problems beyond NP from Papers IV and V. To conclude the thesis, we discuss interesting directions for further work in Chapter 8.

Chapter 2

Preliminaries

In this chapter we formalize the concepts of hitting sets and implicit hitting set algorithms on which the contributions of this thesis build. We also review concepts related to Boolean satisfiability (SAT) and integer programming (IP), which are useful for understanding the use of SAT and IP solvers in the rest of the thesis. We begin with preliminaries on SAT and IP followed by a formal definition of the (explicit) hitting set problem.

2.1 Boolean Satisfiability

Boolean satisfiability (SAT) [63] is the classical NP-complete problem [40] of determining whether there exists a truth assignment that satisfies a given propositional formula. Without loss of generality we consider only formulas in conjunctive normal form (CNF). Arbitrary propositional formulas can be converted to CNF by basic rules of Boolean algebra [127], though this can result in an exponential increase in the size of the formula. However, the well-known Tseitin transformation [152] introduces at most a linear number of new variables and clauses. The structure of CNF formulas is simple.

- A literal l is a Boolean variable or its negation, x or $\neg x$.
- A clause C of length m is a disjunction of literals, $l_1 \vee \cdots \vee l_m$.
- A formula F with k clauses is a conjunction $C_1 \wedge \cdots \wedge C_k$.

Where convenient we will treat a CNF formula F as the set of its clauses and a clause C as the set of its literals.

A complete *truth assignment* $\tau : X \to \{0,1\}^n$ assigns a value of 0 or 1 to each variable $x_1, \ldots, x_n$. For ease of notation, we lift $\tau(\cdot)$ to literals, clauses, and formulas.

- A literal l is satisfied, $\tau(l) = 1$, if l is a positive literal x and $\tau(x) = 1$ or if l is a negative literal $\neg x$ and $\tau(x) = 0$.

- A clause C is satisfied, $\tau(C) = 1$, if at least one of its literals is satisfied.

- A formula F is satisfied, $\tau(F) = 1$, if all of its clauses are satisfied.

If τ is a satisfying truth assignment for F we may also say that (the set of literals satisfied by) τ is a model of F, denoted by $\tau \models F$. A formula F is *satisfiable* if a truth assignment τ for which $\tau(F) = 1$ exists.

Computer programs which take as input a CNF formula F and search for a satisfying assignment are known as SAT solvers [109]. A complete solver is guaranteed to find a solution in finite time if one exists while an incomplete solver is not. Many complete solvers also produce a proof of unsatisfiability if no solution exists [71, 157]. Driven by yearly competitions [90], SAT solvers have become efficient tools for attacking encodings of large-scale real-world problems. SAT solvers are commonly used in a "model-and-solve" workflow, using CNF formulas as a declarative programming language. In general terms, suppose that we are given a problem instance I and we want to find a solution $soln(I)$ of I. Rather than developing an algorithm to solve instances of this problem, it can be more convenient to model the problem in propositional logic and convert I to a CNF formula F_I such that satisfying assignments τ to F_I correspond to solutions of I. Such a τ can be found using an off-the-shelf SAT solver and interpreted as a solution to I. Figure 2.1 visualizes this process.

Variations of the conflict-driven clause learning (CDCL) algorithm [109] represent the current state of the art in complete solvers for non-random SAT instances. A long line of solvers, including GRASP [110], Chaff [116], MiniSat [52], Glucose [15], and Lingeling [30], have refined heuristics and search techniques for CDCL. The technical details of implementing SAT solvers are beyond the scope of this thesis. We utilize them in essentially a black-box manner within optimization procedures both for determining the satisfiability of a formula and producing a truth assignment (if satisfiable) or isolating an unsatisfiable subset of clauses (if unsatisfiable).

$$I \xrightarrow{model} F_I \xrightarrow{solve} \tau \models F_I \xrightarrow{interpret} soln(I)$$

Figure 2.1: The model and solve workflow.

2.2 Integer Programming

An integer programming (IP) [142] problem over n variables $x_1, \ldots, x_n$ consists of a linear objective function, m linear constraints (inequalities), and integrality constraints. The goal is to maximize (or minimize) an objective function

$$f(\bar{x}) = w_1 x_1 + \cdots + w_n x_n$$

with coefficients $w_1, \ldots, w_n$, subject to linear constraints of the form

$$a_{j1} x_1 + \cdots + a_{jn} x_n \leq b_j \text{ for } 1 \leq j \leq m,$$

often written as $A\mathbf{x} \leq \mathbf{b}$. Unlike linear programs, (mixed) integer programs also constrain (a subset) of the variables to integer values. In the applications we consider in this thesis, the variables are often binary,

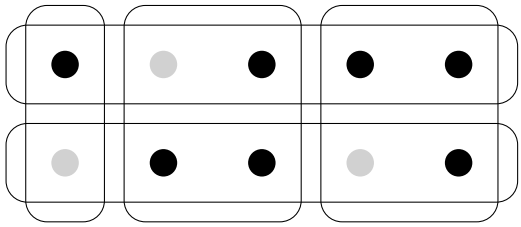
$$x_i \in \{0, 1\} \text{ for } 1 \leq i \leq n.$$

Integer programming with binary variables (also known as 0-1 integer programming) is NP-complete [94]. However, the linear programming (LP) relaxation obtained by removing the integrality constraints can be solved in polynomial time [95]. In this thesis, we use these LP relaxations to obtain lower bounds for IP problems. Integer programming is widely used in areas of operations research including planning, scheduling, and network optimization. These problems are not only of theoretical interest, commercial solvers such as IBM CPLEX [84] and Gurobi [74] are heavily optimized for practical applications.

2.3 Hitting Sets

Given a universe of elements $U = \{e_1, \ldots, e_m\}$ and a collection of subsets of these elements $\mathcal{S} = \{s_1, \ldots, s_n : s_i \subseteq U\}$, a *hitting set* H of $\mathcal{S}$ includes at least one element of each s_i. In other words $H \cap s_i \neq \emptyset$ for all $s_i \in \mathcal{S}$. We use $\mathcal{H}$ to denote the collection of all hitting sets of $\mathcal{S}$. We can assume all supersets of any $s_i \in \mathcal{S}$ to also be in $\mathcal{S}$ as this does not change $\mathcal{H}$. The hitting set problem we consider is an equivalent reformulation of the set cover problem, so it is NP-complete to determine if a hitting set of size k exists [94]. Note that this problem setting differs from the hitting set problem of Karp's 21 NP-complete problems [94], which instead asks if there exists a set H such that $|H \cap s_i| = 1$ for all $1 \leq i \leq n$.

Example 2.1. In the following figure, let U be the set of filled circles and $\mathcal{S}$ the boxes. The gray circles represent a hitting set of $\mathcal{S}$.



Commonly the elements of $\mathcal{S}$ represent subsets of U that are in some sense inconsistent. A subset of U which is not a superset of any $s \in \mathcal{S}$ is then *consistent*. Consistent subsets of U and elements of $\mathcal{S}$ are linked by hitting sets. The property of hitting set duality [31,47,131,132] states that

1. a hitting set of $\mathcal{S}$ is the complement of a consistent subset of U, and conversely
2. a hitting set of $\mathcal{H}$ is a superset of a set in $\mathcal{S}$.

A similar property holds for minimal hitting sets. In case (1), if $U \setminus H$ is not consistent, there exists $s \in \mathcal{S}$ such that $s \subseteq U \setminus H$, so H is not a hitting set of $\mathcal{S}$. The reasoning for case (2) is essentially symmetric. This simple property is the basis for the implicit hitting set algorithms considered in this thesis.

In the context of optimization, the *minimum hitting set* problem is to find a smallest (by cardinality) hitting set in $\mathcal{H}$. Generalizing the problem, costs or weights can be assigned to each $e \in U$ by a function $c : U \to \mathbb{N}$. Extending the cost function to $H \subseteq U$, $c(H) = \sum_{e \in H} c(e)$. An instance of this *minimum-cost hitting set* (MCHS) problem is a tuple $(U, \mathcal{S}, c)$ where the goal is to find a hitting set $\hat{H}$ of $\mathcal{S}$ of smallest cost,

$$\hat{H} \in \operatorname*{arg\,min}_{H \in \mathcal{H}} c(H).$$

A standard method for solving the MCHS problem is to reformulate it as an integer program

$$\begin{aligned} \text{Minimize } & \sum_{i=0}^{m} x_i \cdot c(e_i) \\ \text{Subject to } & \sum_{e_i \in s_j} x_i \geq 1 \text{ for all } s_j \in \mathcal{S}, \\ & x_i \in \{0,1\} \text{ for all } e_i \in U. \end{aligned} \tag{2.1}$$

In words, the hitting set IP introduces for each element e_i in U a binary selector variable. Constraints for each $s_j \in \mathcal{S}$ state that at least one element in each s_j is selected, enforcing that the true selector variables of a solution to the IP correspond to a hitting set of $\mathcal{S}$. The objective function minimizes the total weight of selected elements.

In later chapters we see cases where it is enough to compute a "good enough" hitting set instead of an optimal one. When an approximation is sufficient, Chvátal's greedy heuristic for weighted set-covering [36] can be adapted to the hitting set problem. This simple polynomial-time procedure produces a hitting set with cost within about a $\ln n$ factor of optimal.

Chapter 3

The Implicit Hitting Set Approach

In this section we recall the implicit hitting set approach on which the contributions of this thesis build. Following Paper IV, we formally define the implicit hitting set problem and a generic algorithm for solving it. We also give an overview of previously proposed heuristic search techniques for improving the algorithm.

3.1 Implicit Hitting Sets

In contrast to the (explicit) hitting set problem, in an implicit hitting set problem the collection of subsets $\mathcal{S}$ to hit is either not known or not directly accessible. Instead, $\mathcal{S}$ is implicitly expressed by $p : \mathcal{P}(U) \to \{true, false\}$, a predicate which tests the feasibility of a set $H \subseteq U$. Given a hitting set candidate H, $p(H) = true$ if and only if H hits every element of $\mathcal{S}$.

Implicit hitting set problems often stem from explicit hitting set problems where it is intractable to enumerate $\mathcal{S}$ due to its size or the difficulty of computing its elements, or both. An implicit hitting set algorithm aims to find the optimal hitting set without direct knowledge of every element of $\mathcal{S}$. We use an iterative approach for solving these problems, which alternates between constructing hitting set candidates (underapproximations) H and evaluating $p(H)$.

The applications of Papers I–V solve implicit hitting set problems in different contexts. In each case U is a set of constraints, with syntax and semantics depending on the application. To make our terminology consistent with these applications, we will call members of $\mathcal{S}$ cores. In later chapters we call constraints over U which are not in $\mathcal{S}$ non-core constraints [45] and hitting sets computed for some $S' \subseteq \mathcal{S}$ which are not minimum-cost non-optimal hitting sets.

3.2 Generic Algorithm

Implicit hitting set algorithms were first proposed as a generic optimization framework by Chandrasekaran et al. [34] and Moreno-Centeno and Karp [113]. Their approach was restricted to problems where computing $p(H)$ and extracting a core is possible in polynomial time. The approach we take in Paper IV lifts the framework to problems beyond NP by considering arbitrarily hard predicates p and core extraction procedures.

Algorithm 1 outlines this generic solution to implicit hitting set problems. It either returns a minimum-cost hitting set of $\mathcal{S}$ if one exists (Line 6) or reports that no solutions exist (Line 10) in the special case that $\emptyset \in \mathcal{S}$. The algorithm maintains a subset of identified cores $S' \subseteq \mathcal{S}$, initially an empty set, and a hitting set candidate H, also initially an empty set.

As long as $p(H) = \mathit{false}$, the algorithm will identify a new core and add it to S' (Line 8). An MCHS-oracle (e.g. an IP solver) is then used to find a minimum-cost hitting set for S' and update H (Line 9). New cores are identified with the *extractcore* procedure. The implementation of this procedure may vary by application. In the most generic case we can set $\mathit{extractcore}(H) = U \setminus H$ because if H is not a hitting set of $\mathcal{S}$, then every hitting set of $\mathcal{S}$ must contain an element of $U \setminus H$. Depending on the application, the functions of p and *extractcore* can be accomplished by the same procedure. For example in the case of MaxSAT in Chapter 4, a SAT solver produces an unsatisfiable core if the predicate check fails.

When $p(H) = \mathit{true}$ on Line 5, H is by construction a minimum-cost hitting set for $\mathcal{S}$. It is not immediately obvious that Algorithm 1 is correct in that it produces an optimal solution and halts for all instances of the implicit hitting set problem. To formalize the correctness of the algorithm we

Algorithm 1 A generic implicit hitting set algorithm.

```
1:  procedure IHS(p, c)
2:      H ← ∅
3:      S' ← ∅
4:      while ∅ ∉ S' do
5:          if p(H) then
6:              return H
7:          else
8:              S' ← S' ∪ {extractcore(H)}
9:          H ← MCHS(S', c)
10:     return "no solution"
```

adapt the proof of Davies and Bacchus, presented originally in the context of MaxSAT [44].

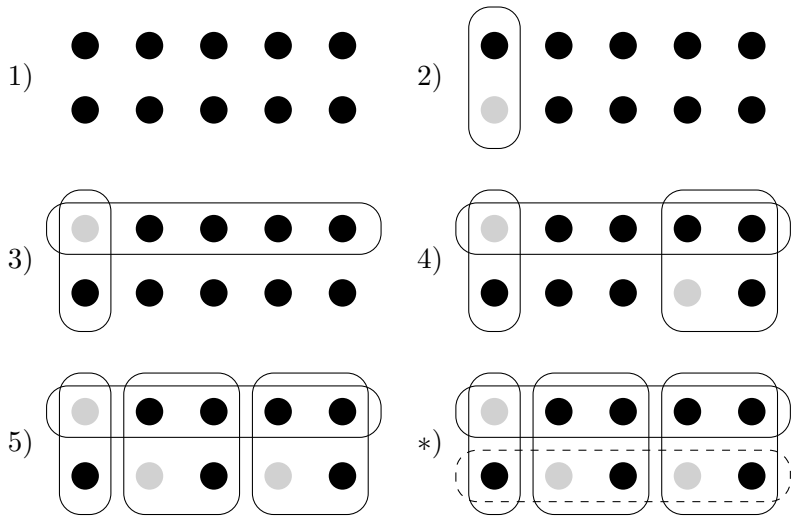
Theorem 3.1. *Algorithm 1 computes a minimum-cost hitting set of $\mathcal{S}$ and halts.*

Proof. Firstly, Algorithm 1 returns H only if it is feasible (i.e. $p(H)$ is true because it hits every core of $\mathcal{S}$). The returned set H is a MCHS for some $S' \subseteq \mathcal{S}$. It must then also be a MCHS for $\mathcal{S}$.

Secondly, after each iteration of the loop, H intersects (at least) the cores of $\mathcal{S}$ which are in S'. On each iteration, the algorithm either terminates or finds $s \in \mathcal{S}$ not hit by H and adds it to S'. Since H hits each core of S', this core s must be distinct from all cores previously added to S'. The algorithm must terminate since U and thus also $\mathcal{S}$ is finite. □

The generic implicit hitting set algorithm gives us a powerful framework for solving hard optimization problems. In this thesis we instantiate the algorithm for several constraint optimization settings. In principle Algorithm 1 could be instantiated for any constraint language, given the predicate p and a core extraction procedure.

Example 3.1. Continuing in the (unweighted) setting of Example 2.1, we illustrate a sequence of steps which could be taken by Algorithm 1. At each step, the gray circles represent H and the black boxes represent S'. At step 5, the algorithm has found an optimal hitting set and can terminate. The last step shows a core which is implicitly hit.



3.3 Heuristics for IHS

In the introduction we gave a short overview of applications of the implicit hitting set approach. A number of these applications introduced useful heuristics which are applicable to implicit hitting set algorithms and the generic setting of Algorithm 1. Though they do not improve on the theoretical worst-case running time bounds, these heuristics cut down on the number of times the predicate p must be checked or the number of calls to the *extractcore* procedure or MCHS solver required in practice.

The work of Parker and Ryan [121], which uses an implicit hitting set algorithm to compute maximum feasible subsystems, suggests the use of a greedy hitting set algorithm [36] and finding an initial disjoint set of cores (irreducible inconsistent subsystems in the original context). They note that the hitting set problem can be solved heuristically at intermediate steps of the algorithm and use the well-known greedy approximation for this task. To ensure an optimal solution it is only necessary to solve the hitting set problem exactly at the last iteration. They also observe that the initial disjoint set of cores has the benefit of producing a lower bound on the size of the optimal hitting set.

Davies and Bacchus [44] make extensive use of heuristics in their implicit hitting set algorithm for MaxSAT. They implement a *core minimization* procedure to reduce the size of the cores passed to the hitting set solver. Giving the hitting set solver a core $s \in \mathcal{S}$ that is not subset-minimal with respect to inclusion in $\mathcal{S}$ can cause the implicit hitting set algorithm to require more interations to terminate. However, finding a minimal core (i.e. MUS extraction [22, 108]) requires more computation by the *extractcore* procedure. Davies and Bacchus [44] also note the importance of *core diversity*. Simply put, the implicit hitting set algorithm will tend to require fewer iterations when the core uncovered at each iteration is not similar to cores found on previous iterations. Motivated by increasing core diversity, the MaxHS algorithm introduces randomness to the SAT-oracle and performs an initial disjoint phase. Subsequent work on the MaxHS algorithm introduced yet more search heuristics, including the use of non-core constraints [45, 46].

The minimum-cost hitting set problem is typically solved using an IP solver so search techniques developed for integer programs can often find use in implicit hitting set algorithms. For example, Moreno-Centeno and Karp [113] use the LP relaxation of the hitting set IP to compute lower bounds. Reduced-cost fixing techniques [41, 42] have also been applied to implicit hitting set algorithms [16, 17]. We discuss some IP and LP based techniques in Chapters 5 and 7.

Chapter 4

Maximum Satisfiability

In this chapter we discuss the contributions of this thesis to solving maximum satisfiability (MaxSAT) presented in Papers I and II. Following the rapid development of efficient practical algorithms for Boolean satisfiability (SAT), algorithms for MaxSAT which make use of these SAT procedures are increasingly utilized for real-world optimization tasks. Recent practical applications of MaxSAT include several data analysis problems [25], model-based diagnosis [106], data visualization [32], hardware and software error localization [93, 158], and scheduling problems [49]. Our contribution is towards improving the generic MaxSAT solving procedures behind these practical applications.

We begin with a discussion of MaxSAT to the extent necessary to cover the contributions of Paper I and Paper II. An overview of the LMHS MaxSAT solver presented in Paper I follows. The implicit hitting set algorithm for MaxSAT was first implemented in the MaxHS solver [43]. We discuss the differentiating features between our solver and MaxHS. In the latter half of the chapter we take closer look at Paper II and one of the key features of LMHS, the integration of preprocessing.

An implicit hitting set based algorithm is an attractive target for MaxSAT preprocessing because it does not apply transformations which increase the size of the formula during search. Intuitively, we expect this to make the benefits of preprocessing simplifications to the formula last longer. The use of SAT-based preprocessing techniques for MaxSAT is made possible by the labeled CNF (LCNF) framework [23, 24]. We show how these techniques can be applied more efficiently by instantiating the implicit hitting set algorithm directly for LCNF formulas. A summary of our experimental evaluation on the LMHS solver and subsequent related work concludes the chapter.

4.1 Maximum Satisfiability

Given an unsatisfiable formula, it is natural to ask what is the largest subset of its clauses that can be simultaneously satisfied. This is the problem of maximum satisfiability (MaxSAT) [101]. More formally, we want to find a truth assignment $\hat{\tau}$ that satisfies as many clauses of a CNF formula F as possible,

$$\hat{\tau} \in \arg\max_{\tau} \sum_{C \in F} \tau(C).$$

In practical applications there is often a need to enforce hard constraints on the solution space or designate weights to clauses to prioritize them. Weighted partial MaxSAT combines these generalizations of the MaxSAT problem. Formally, an instance of the weighted partial MaxSAT problem is a tuple (F_h, F_s, w) where F_h is a set of mandatory *hard* clauses, F_s a set of optional *soft* clauses, and $w : F_s \to \mathbb{N}$ assigns a positive weight to each soft clause. The weight of a set of soft clauses $S \subseteq F_s$ is the sum of the weights of the clauses in the set, $w(S) = \sum_{C \in F_s} \tau(C)$. Plain MaxSAT is then a special case of weighted partial MaxSAT in which $F_h = \emptyset$ and $w(C) = 1$ for all $C \in F_s$. The weighted partial generalization of MaxSAT sees so much use that we will refer to it as MaxSAT from hereon.

Example 4.1. The following is a weighted partial MaxSAT instance.

$$F_h = \{(x_1 \vee x_2), (\neg x_1 \vee \neg x_2), (\neg x_1 \vee x_2 \vee \neg x_3), (x_1 \vee \neg x_2 \vee \neg x_4)\}$$
$$F_s = \{(x_1, 5), (x_2, 7), (x_3, 1), (x_4, 4)\}$$

Hard clauses $(x_1 \vee x_2)$, $(\neg x_1 \vee \neg x_2)$, $(\neg x_1 \vee x_2 \vee \neg x_3)$, and $(x_1 \vee \neg x_2 \vee \neg x_4)$ must be satisfied and the cost function w assigns weights 5, 7, 1 and 4 to soft clauses (x_1), (x_2), (x_3), (x_4), respectively.

Extending the semantics of CNF formulas, a truth assignment τ is a *solution* to a MaxSAT instance (F_h, F_s, w) if $\tau(F_h) = 1$. Further, $\hat{\tau}$ is optimal if the sum of the weights of soft clauses it satisfies is maximal,

$$\hat{\tau} \in \arg\max_{\tau \models F_h} \sum_{C \in F_s} \tau(C) w(C).$$

We say the cost of a MaxSAT solution τ is the sum of weights of soft clauses not satisfied by τ,

$$cost(\tau) = \sum_{C \in F_s} (1 - \tau(C)) w(C).$$

Example 4.2. Continuing with the formula of Example 4.1,

$$\tau_1 = \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0\}$$

is not a MaxSAT solution because $\tau_1((x_1 \vee x_2)) = 0$. The truth assignment

$$\tau_2 = \{x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0\}$$

is a solution with cost 9, as it satisfies all soft clauses except (x_1) and (x_4) which have weights 5 and 4, respectively. However, τ_2 is not optimal because

$$\tau_3 = \{x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1\}$$

is also a solution and has a cost of 8.

An unsatisfiable *core* of a MaxSAT instance (F_h, F_s, w) is a subset of soft clauses $\kappa \subset F_s$ such that $F_h \cup \kappa$ is not satisfiable. A core κ is minimal if no subset of it is a core, that is, $F_h \cup \kappa'$ is satisfiable for all $\kappa' \subsetneq \kappa$. Minimal cores are also known as minimal unsatisfiable subsets, or MUSes.

Example 4.3. Taking again the formula (F_h, F_s, w) of Example 4.1, the set of soft clauses $\{(x_1, 5), (x_2, 7), (x_3, 1)\}$ is a core of F, but not a minimal core, since $F_h \cup \{(x_1, 5), (x_2, 7)\}$ is also unsatisfiable. In all, F has minimal cores

$$\big\{\{(x_1, 5), (x_2, 7)\}, \{(x_1, 5), (x_3, 1)\}, \{(x_3, 1), (x_4, 4)\}, \{(x_2, 7), (x_4, 4)\}\big\}.$$

In practice, cores can be identified using a SAT solver which can solve with assumptions [14, 52]. These solvers can take as input, in addition to a CNF formula F, a set of *assumption literals* A. The solver will then search for a satisfying assignment to F which also satisfies the given assumptions. If no such assignment exists, the solver can produce a subset of A which was used in proving unsatisfiability. It is common practice to find a core of a MaxSAT formula (F_h, F_s, w) by augmenting each soft clause $C_i \in F_s$ with a new, unique assumption variable a_i to produce augmented soft clauses

$$F_s^a = \{(C_i \vee a_i) : C_i \in F_s\}.$$

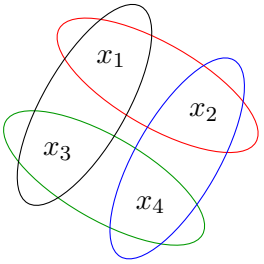
Solving $F_h \cup F_s^a$ with assumptions $A = \{\neg a_i : C_i \in F_s\}$ is then analogous to solving $F_h \cup F_s$ with no assumptions. Due to the one-to-one relationship between assumption variables and soft clauses, the subset of assumptions used to prove unsatisfiability will correspond to a core of the MaxSAT formula.

4.2 Instantiating IHS for MaxSAT

The implicit hitting set algorithm can be instantiated for MaxSAT using the concept of unsatisfiable cores. For the following, consider a MaxSAT instance $F = (F_h, F_s, w)$ and suppose that F_h is satisfiable and $F_h \cup F_s$ is unsatisfiable.

Davies and Bacchus [44] explore an important relationship between hitting sets and MaxSAT solutions. Let $\mathcal{K}$ be the set of all cores of F. Every hitting set H of $\mathcal{K}$ corresponds to at least one MaxSAT solution of F, the truth assignment(s) which satisfy $F_h \cup (F_s \setminus H)$. Conversely, every MaxSAT solution τ corresponds to a hitting set of $\mathcal{K}$, the set of soft clauses not satisfied by τ. Moreover, if $\tau(F_h \cup (F_s \setminus H)) = 1$, then $cost(\tau) \leq w(H)$ and if H is minimal, $cost(\tau) = w(H)$ [44]. It follows that an optimal MaxSAT solution can be computed by first finding a minimum-weight hitting set $\hat{H}$ of $\mathcal{K}$ and then solving the satisfiable CNF formula $F_h \cup (F_s \setminus \hat{H})$.

Example 4.4. The cores of Example 4.3 overlap as follows, illustrating the structure of the underlying hitting set problem.



Motivating the use of the IHS approach, there are cases in which it is impractical to solve the hitting set problem directly because a MaxSAT instance can have an exponential number of minimal cores. Using an example of Davies and Bacchus [46] suppose that $F_s = \{(x_i) : 1 \leq i \leq m\}$ and F_h is a CNF encoding of the cardinality constraint $\sum_{i=1}^{m} x_i < \frac{m}{2}$. Now if an assignment τ satisfies exactly $m/2$ soft clauses, then it does not satisfy F_h so any set of $m/2$ soft clauses is a (minimal) core. There are $\binom{m}{m/2}$ such cores. However, the implicit hitting set algorithm gives us a way to find the optimal solution without necessarily enumerating all cores.

Davies and Bacchus [43–46] first proposed the MaxHS algorithm, which instantiates Algorithm 1 for MaxSAT. The structure of the algorithm is loosely depicted in Figure 4.1. Initially the clauses of the MaxSAT instance are input to the SAT solver and the cost function is input to the IP solver. The algorithm alternates between calls to a SAT solver and calls to an IP solver. The SAT solver attempts to satisfy the hard clauses F_h and the

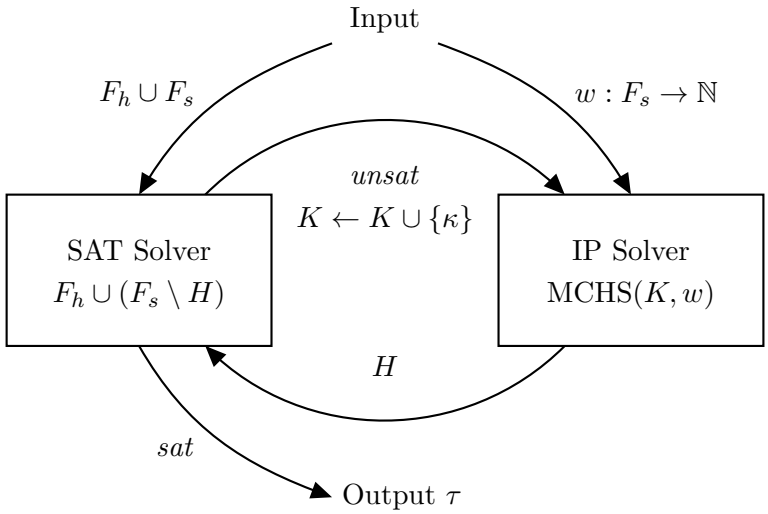


Figure 4.1: The implicit hitting set algorithm for MaxSAT.

soft clauses not in the current hitting set $F_s \setminus H$. If the set of clauses is unsatisfiable, a core κ is found by the SAT solver and the IP solver computes a new optimal hitting set. Otherwise a satisfying assignment is found and the algorithm terminates. By Theorem 3.1 and the earlier observation that a hitting set corresponds to a solution with the same cost (and vice versa) this solution will be optimal.

Example 4.5. Consider again the formula of Example 4.1,

$$F_h = \{(x_1 \vee x_2), (\neg x_1 \vee \neg x_2), (\neg x_1 \vee x_2 \vee \neg x_3), (x_1 \vee \neg x_2 \vee \neg x_4)\},$$
$$F_s = \{(x_1, 5), (x_2, 7), (x_3, 1), (x_4, 4)\}.$$

On the first iteration the implicit hitting set algorithm will begin with an empty hitting set $H_0 = \emptyset$ and thus try to satisfy $F_h \cup F_s$. This formula is unsatisfiable so the SAT solver will identify a core, say $\kappa_0 = \{(x_1, 5), (x_2, 7)\}$. The minimum-cost hitting set for the single core is clearly $H_1 = \{(x_1, 5)\}$ with cost $c(H_1) = 5$ so the algorithm will next try to satisfy $F_h \cup (F_s \setminus H_1)$. In all, the algorithm might consider the following sequence of cores and hitting sets.

i	K	H	$c(H)$	κ
0	$\emptyset$	$\emptyset$	0	$\{x_1, x_2\}$
1	$\{\{x_1, x_2\}\}$	$\{x_1\}$	5	$\{x_3, x_4\}$
2	$\{\{x_1, x_2\}, \{x_3, x_4\}\}$	$\{x_1, x_3\}$	6	$\{x_2, x_4\}$
3	$\{\{x_1, x_2\}, \{x_3, x_4\}, \{x_2, x_4\}\}$	$\{x_2, x_3\}$	8	$\emptyset$

On the final iteration $i = 3$, $F_h \cup (F_s \setminus H_3)$ is satisfiable. This means that H_3 is a minimum-cost hitting set for all cores of (F_h, F_s, w), so truth assignments which satisfy $F_h \cup (F_s \setminus H_3)$ are optimal MaxSAT solutions for (F_h, F_s, w). In this case the optimal solution is unique,

$$\hat{\tau} = \{x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1\}.$$

The algorithm arrives at the solution without explicitly considering every core of (F_h, F_s, w). The core $\{x_1, x_3\}$ is implicitly hit by H_3.

4.3 LMHS

Our MaxSAT solver LMHS, a from-scratch implementation of the implicit hitting set algorithm, is introduced in Paper I. LMHS set itself apart in several ways from the original implementation, MaxHS, including two key differences in search techniques. The solver integrated novel preprocessing techniques based on the so-called labeled CNF (LCNF) framework [23, 24]. Preprocessing for MaxSAT is the topic of Paper II, the contributions of which are covered in the next section.

LMHS also boosts core diversity by computing a disjoint set of cores at *every* iteration of the solver, a technique later adapted to MaxHS as well. This is in contrast to the initial disjoint phase seen in other core-based algorithms, such as msu4 [107]. This simple heuristic was especially effective on industrial instances, enabling LMHS to solve the greatest number of instances in the highly competitive class of complete solvers of the weighted partial category of the 2015 MaxSAT evaluation. The results of the evaluation are shown in Figure 4.2.

The solver includes a number of usability features not related to the search algorithm itself.

Solution enumeration Simple command-line parameters enable MaxSAT solution enumeration. LMHS can enumerate all solutions, all optimal solutions, or a single solution for each unique set of satisfied soft clauses.

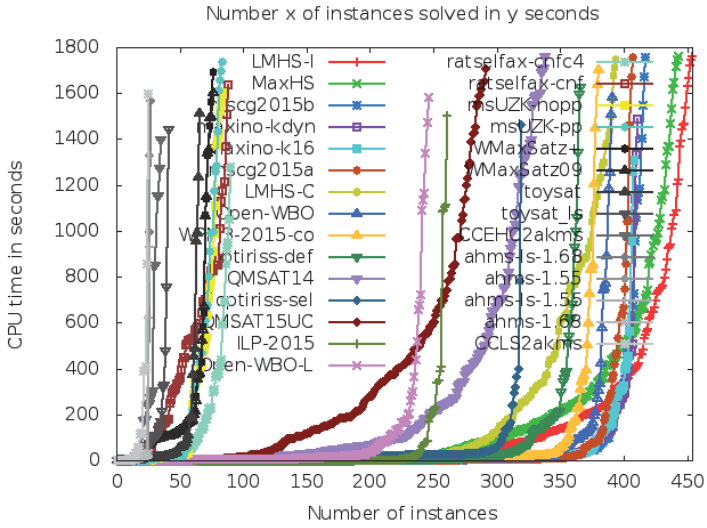


Figure 4.2: Industrial category results of the 2015 MaxSAT Evaluation [12].

C++ API LMHS can also be used through its C++ library and API. Hard and soft constraints as well as cores can be added to the working formula incrementally. This functionality was used for example to compute cutting planes within an IP-based procedure for learning optimal Bayesian networks [137].

Interchangeable components The robustness of LMHS was increased by the ability to change the IP and SAT components used by the solver. The original release of LMHS supported a choice of IBM CPLEX [84] or SCIP [2] for solving the hitting set IP and a choice of MiniSat [52] or Lingeling [30] as a SAT solver.

4.4 Preprocessing

Preprocessing and inprocessing techniques have long been an integral part of SAT solvers [51, 91, 92]. We use preprocessing to mean the application of reversible satisfiability-preserving polynomial-time transformations to a CNF formula to simplify it prior to solving. Figure 4.3 shows the place of preprocessing in the model-and-solve workflow (recall Figure 2.1). Preprocessing steps are applied to the CNF formula F_I transforming it into an equisatisfiable formula F_I'. If a satisfying assignment τ' is found for F_I' a satisfying assignment τ for F_I can be reconstructed from it based on the preprocessing steps applied to F_I.

$$I \xrightarrow{model} F_I \xrightarrow{preprocess} F_I' \xrightarrow{solve} \tau' \models F_I' \xrightarrow{reconstruct} \tau \models F_I \xrightarrow{interpret} soln(I)$$

Figure 4.3: Preprocessing in the model-and-solve workflow.

By comparison, the potential of preprocessing techniques for MaxSAT has been less thoroughly explored. Paper II builds on recent work [23, 24] to refine the use of well-known SAT preprocessing techniques of blocked clause elimination (BCE) [91], bounded variable elimination (VE) [51], self-subsuming resolution (SSR) [51], and subsumption elimination (SE) with the implicit hitting set algorithm for MaxSAT. We investigate the use of these techniques in conjunction with the LMHS solver of Paper I.

4.4.1 Labeled CNF formulas

Though SAT-based preprocessing techniques preserve satisfiability, they cannot be used directly for MaxSAT [24] because they do not necessarily preserve the set of optimal solutions of a formula. The labeled CNF (LCNF) framework [23, 24] has been proposed to remedy this problem.

A LCNF formula Φ is a set of pairs (C, L) where C is a clause and $L \subset \mathbb{N}$ is a finite set of labels. We will use C^L as shorthand for a labeled clause (C, L) and denote the sets of clauses and labels of Φ with

$$Cl(\Phi) = \{C : C^L \in \Phi\}$$

and

$$Lbls(\Phi) = \bigcup_{C^L \in \Phi} L,$$

respectively. A LCNF formula Φ is satisfiable if $Cl(\Phi)$ is satisfiable. Given a LCNF formula Φ, a set of labels $M \subseteq Lbls(\Phi)$ induces the subformula

$$\Phi|_M = \{C^L \in \Phi : L \subseteq M\}.$$

From a practical perspective, labels can be seen as a generalization of the assumptions used in iterative SAT solving. Inducing Φ by a label set which does not include a label l corresponds to removing from Φ each clause associated with l. An unsatisfiable core $\kappa \subset Lbls(\Phi)$ of Φ is a label-set such that $\Phi|_\kappa$ is unsatisfiable, while a correction subset of Φ is a set of labels $R \subset Lbls(\Phi)$ such that $\Phi|_{Lbls(\Phi)\setminus R}$ is satisfiable. If we additionally associate each label $l \in Lbls(\Phi)$ with a weight (or cost) $c(l)$, we can define weighted maximum LCNF-satisfiability as the optimization problem of finding a minimum-cost correction subset (MCS) of Φ.

4.4.2 Applying SAT-based Preprocessing

The LCNF framework provides a way to apply SAT-based preprocessing techniques for MaxSAT while preserving the cost of optimal solutions. Given a MaxSAT instance (F_h, F_s, w),

1. lift the MaxSAT instance to a LCNF Φ,
2. apply LCNF-liftings of preprocessing techniques to Φ, transforming it into a LCNF Φ', and
3. convert Φ' back to a MaxSAT instance.

An explanation of each step follows. A MaxSAT instance (F_h, F_s, w) can be lifted to maximum LCNF-satisfiability by a simple transformation. First, each soft clause $C_i \in F_s$ is assigned a label l_i with weight $c(l_i) = c(C_i)$. Then, an equivalent LCNF instance Φ can be formed from the labeled clauses $\{C^{\emptyset} : C \in F_h\} \cup \{C_i^{l_i} : C_i \in F_s\}$. Hard clauses of the MaxSAT instance are by definition present in an induced subformula $\Phi|_M$ for any $M \subseteq Lbls(\Phi)$ because they are given an empty label set.

The preprocessing techniques of VE, SSR, and SE can be adapted, or lifted, to LCNF formulas. The lifted techniques and associated proofs of correctness for these techniques were presented in Belov et al. [24]. Berg and Järvisalo [27] later presented a general theorem of correctness for applying SAT-based preprocessing. Informally, while the lifted techniques eliminate variables or clauses from the formula, the structure of *labels* is preserved. This allows the lifted preprocessing techniques to preserve the MUSes and MCSes of a formula, a sufficient condition for the preprocessing technique to preserve optimal solutions [24].

A labeled CNF can be easily converted to a MaxSAT instance by a direct encoding [24] which introduces a new soft clause for each label. In this way LCNF formulas can be solved by any off-the-shelf MaxSAT solver, but it has been observed that introducing the new soft clauses can have a negative impact on search [24].

The implicit hitting set algorithm can instead be instantiated for LCNF, with subsets of labels instead of soft clauses as cores. In practice, the core extraction process described in Section 4.1 for MaxSAT can be adapted to LCNF formulas by using labels as assumptions. After preprocessing, many clauses can be covered by the same label, making these label assumptions possibly more powerful than the assumptions used for MaxSAT.

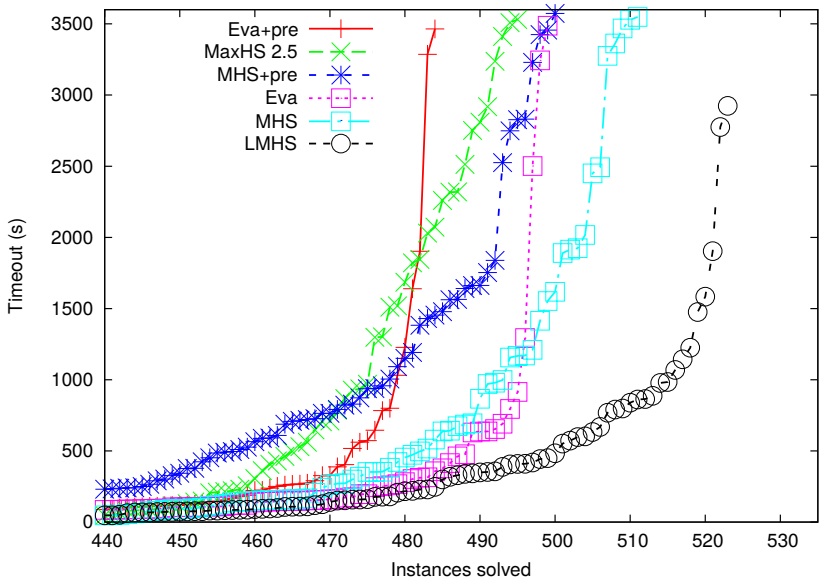


Figure 4.4: Impact of preprocessing on solving times.

4.4.3 Experimental Evaluation

Paper II includes a thorough experimental evaluation on the use of preprocessing with IHS and core-guided MaxSAT solvers. We report here on the aspects most relevant to this thesis. The impact of preprocessing, evaluated on all 624 weighted partial instances [11] of the 2014 MaxSAT evaluation, is shown in Figure 4.4. We see that for both a core-guided solver (Eva) [117] and our implementation of the implicit hitting set algorithm (MHS), the direct application of preprocessing (Eva+pre and MHS+pre) is detrimental to the performance of the solver. For reference the results are also compared against MaxHS 2.5, (at the time of writing Paper II) the latest version of the IHS-based MaxHS solver [43]. LMHS performs preprocessing and solving within the LCNF framework. In contrast to the direct application of preprocessing (MHS+pre), we see a significant increase in the number of instances solved compared to MHS.

4.4.4 Related Work

The work reported on in Paper II was directly continued in [28] and [29]. While Paper II efficiently integrated existing SAT-based techniques into a MaxSAT solver, the following work developed novel MaxSAT-specific techniques.

A simple observation showed that label variables and group encodings [77] can be detected prior to applying preprocessing and exploited to great effect [28]. The MaxSAT preprocessing technique of subsumed label elimination was introduced and evaluated in [29]. More formal analysis of the impact of preprocessing on core-guided and hitting set based algorithms was later carried out in [26] and [27].

A dedicated MaxSAT preprocessor, MaxPre [97], was also developed as an extension of the work of Paper II. MaxPre implements common SAT-based preprocessing techniques for MaxSAT as well as label-based MaxSAT specific techniques. It can be used as a standalone preprocessor or integrated into a solver using its API.

Chapter 5

Causal Discovery

This chapter gives an overview of the contributions of Paper III to learning, or discovering, causal structures. We specialize the implicit hitting set algorithm for a causal discovery task by extending the LMHS solver of Paper I. The adaptability of the implicit hitting set approach allows us to develop three problem-specific search techniques to boost the performance of LMHS on these instances.

Graphical models, such as Markov and Bayesian networks [96], play an important role in facilitating various probabilistic modeling tasks. A graphical model represents a way of decomposing a joint probability function into a product of conditional probabilities. They provide a compact representation of joint probability functions and allow for efficient and accurate inference from observations [123, Ch. 1]. In the context of Paper III our goal is to construct an optimal *causal graph* with which we can make predictions under interventions. Some features of causal structures can be inferred from passively observed data [123, Ch. 2], but they can be challenging to identify with a finite amount of data. In our setting this challenge manifests itself as a combinatorial problem of optimally reconciling conflicting independence and dependence constraints.

While inexact algorithms such as constraint-based PC and FCI [126] can solve these problems at the cost of accuracy [38], we aim to find globally optimal solutions. Exact methods for learning Bayesian networks have been proposed via e.g. integer programming [21]. We consider a more general class of causal graphs with latent confounders (unobserved variables) and feedback (cyclic graph structures). These are required for accurately representing real-world phenomena in e.g. biology [134]. Triantafillou and Tsamardinos [151] previously use SAT solvers in a similar setting which does not admit these feedback cycles. The first exact approach for this problem setting was implemented via an ASP encoding [82], we use an

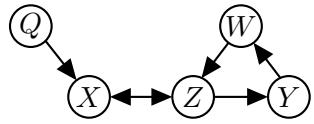
equivalent MaxSAT encoding [25]. In Paper III we further improve on this approach by developing problem-specific search techniques for the implicit hitting set framework.

5.1 Conditional Independence in Causal Graphs

Paper III continues in the problem setting and formulation described by Hyttinen et al. [82]. We consider causal graphs $G = (V, E)$ where the set of nodes V corresponds to a set of (observed) random variables and the edges in E are pairs of nodes representing causal relations between nodes. An edge can be directed ($\rightarrow$) or bi-directed ($\leftrightarrow$). The tail of a directed edge represents the cause. A bi-directed edge $X \leftrightarrow Y$ represents the presence of an unobserved common cause of X and Y such as a structure $X \leftarrow L \rightarrow Y$ where $L \notin V$.

We recall a well-known reachability condition for causal graphs known as *d-separation* (dependence separation) [122, 149]. A *walk* in a causal graph G is any sequence of consecutive edges, allowing repeats. A node X on a walk is a *collider* if consecutive edges on the walk point into the node, i.e. $\rightarrow X \leftarrow$, $\rightarrow X \leftrightarrow$, $\leftrightarrow X \leftarrow$, or $\leftrightarrow X \leftrightarrow$. A walk between X and Y is d-connecting given a conditioning set $S \subseteq V \setminus \{X, Y\}$ if every collider on the walk is in S and every non-collider node on the walk is not in S. A pair of nodes is d-separated given S if no d-connecting walk exists, otherwise they are d-connected.

Example 5.1. Consider the following causal graph of 5 nodes.



The sequence of nodes $Q \rightarrow X \leftrightarrow Z \leftarrow W$ forms a walk in the graph. On this walk, nodes X and Z are colliders.

In the graph X and W are d-connected given $\{Y\}$ through the walk $X \leftrightarrow Z \rightarrow Y \leftarrow Z \leftarrow W$. On the other hand Y and Q are d-separated given $\{W\}$.

We are interested in d-separation because under causal Markov and faithfulness assumptions for G [126, p. 29–42] it is equivalent to statistical independence between random variables. That is, nodes X and Y are d-separated in G given S if and only if random variables X and Y are independent given variables S in data generated by a system with causal

structure G [145]. We use $X \perp\!\!\!\perp Y \mid S$ to mean statistical indepedence and d-separation, and $X \not\perp\!\!\!\perp Y \mid S$ to mean statistical dependence and d-connectivity.

5.2 Causal Discovery as Constraint Optimization

In the causal discovery problem we consider a set of conditional independence and dependence constraints K over V and a function $w : K \to \mathbb{N}$ which assigns weights, or costs, to (in)dependence constraints. Let $\mathcal{G}$ be the set of possible causal structures for V. This causal discovery problem can then be formulated [82] as computing

$$\hat{G} \in \arg\min_{G \in \mathcal{G}} \sum_{\substack{k \in K \\ G \not\models k}} w(k).$$

That is, we wish to find a causal graph $\hat{G}$ which minimizes the weight of (in)dependence constraints k not implied by $(G \not\models k)$ the structure of $\hat{G}$.

The (in)dependence constraints K and their weights w are derived from data sampled from some underlying distribution. Constraints K are obtained via statistical independence tests on the data. We test the dependence or independence of each of $\binom{n}{2}$ pairs of variables conditioned on each of 2^{n-2} subsets of remaining variables for a total of $\binom{n}{2}2^{n-2}$ (in)dependence constraints. These tests can produce errors and the weight $w(k)$ of a constraint k, obtained by local Bayesian model selection [82], can be seen as a measure of the degree of confidence in k. Figure 5.1 illustrates this workflow as a whole. We focus on the final part of this process, solving the MaxSAT-encoded (in)dependence constraints for the optimal graph structure.

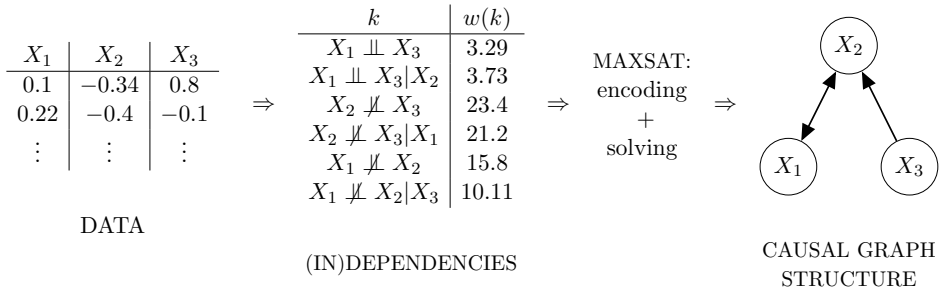


Figure 5.1: Phases of computing a causal graph structure from data [25].

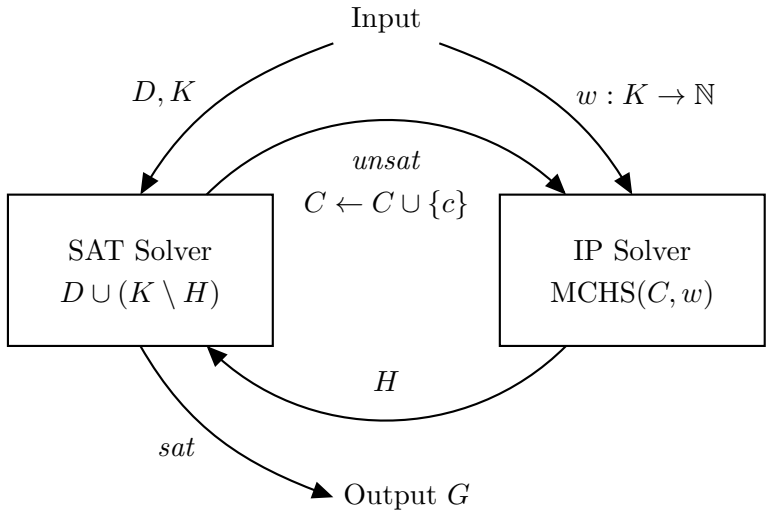


Figure 5.2: The implicit hitting set algorithm of Dseptor.

5.3 Instantiating IHS for Causal Discovery

We translate instances of the causal discovery problem to MaxSAT following earlier work of Hyttinen et al. [25, 82]. The independence and dependence constraints K are represented as soft unit clauses with weight $w(k)$. The d-separation conditions D implied by each (in)dependence constraint are encoded in CNF as hard clauses [25, 82].

We based our solver *Dseptor* on the IHS-based MaxSAT solver LMHS. It was adapted to this domain by building domain-specific improvements to speed up search on causal discovery instances. MiniSat [52] is used as a core extractor and IBM CPLEX [84] is used as an optimizer for the MCHS problems. A one-to-one relationship between the soft clauses and the (in)dependence constraints of the causal structure discovery problem means that cores identified by LMHS will naturally correspond to sets of (in)dependence constraints that cannot simultaneously hold. Figure 5.2 relates the generic implicit hitting set algorithm to the causal discovery setting.

Example 5.2. Consider the set of constraints with unit costs from Paper III:

$$K = \{X \not\perp\!\!\!\perp Z \mid W;\ Y \not\perp\!\!\!\perp Z \mid W;\ X \perp\!\!\!\perp Y \mid W;\ X \perp\!\!\!\perp Y \mid Z, W;$$
$$X \not\perp\!\!\!\perp Z \mid Y, W;\ Y \not\perp\!\!\!\perp Z \mid X, W;\ X \not\perp\!\!\!\perp Y \mid W, Q;\ Y \perp\!\!\!\perp Q \mid W\}.$$

Dseptor first tries to satisfy the entire set of constraints K, and upon finding it unsatisfiable will identify a core, for example

$$c_1 = \{X \not\perp\!\!\!\perp Z \mid W;\ Y \not\perp\!\!\!\perp Z \mid W; X \perp\!\!\!\perp Y \mid W;\ X \perp\!\!\!\perp Y \mid Z, W\}.$$

A minimum-cost hitting set for $\{c_1\}$ will then be computed. One such set is $H_1 = \{X \not\perp\!\!\!\perp Z \mid W\}$ with cost 1. Dseptor will then search for a causal graph satisfying constraints $K \setminus H_1$. However, no such graph exists, so another core

$$c_2 = \{X \perp\!\!\!\perp Y \mid W;\ Y \perp\!\!\!\perp Q \mid W;\ X \not\perp\!\!\!\perp Y \mid W, Q\},$$

is found. The minimum-cost hitting set for $\{c_1, c_2\}$ is $H_2 = \{X \perp\!\!\!\perp Y \mid W\}$, which also has cost 1. There exists a causal graph satisfying the constraints of $K \setminus H_2$ so the SAT solver will find it and stop. This solution (a truth assignment τ satisfying $D \cup (K \setminus H_2)$) has cost 1 and corresponds to the graph of Example 5.1. Here the core

$$c_3 = \{X \not\perp\!\!\!\perp Z \mid Y, W;\ Y \not\perp\!\!\!\perp Z \mid X, W;\ X \perp\!\!\!\perp Y \mid W;\ X \perp\!\!\!\perp Y \mid Z, W\}$$

is not needed for proving optimality.

5.4 Domain-Specific Improvements

The main contribution of Paper III is the addition of three domain specific methods for speeding up search. While the LMHS solver is at the core of Dseptor and the causal discovery problem is in essence translated to Max-SAT, these domain-specific methods significantly improve upon the baseline performance of LMHS by specializing the implicit hitting set approach to the causal graph domain.

Domain-specific cores Seven domain-specific core patterns arising from the d-connectivity conditions were identified. We list here the patterns over three variables. The following are cores for any instantiation of variables A,B,C and set of other variables S:

(i) $\{A \perp\!\!\!\perp C \mid S;\ A \perp\!\!\!\perp B \mid S;\ A \not\perp\!\!\!\perp C \mid B, S\}$

(ii) $\{A \not\perp\!\!\!\perp C \mid S;\ B \not\perp\!\!\!\perp C \mid S;\ A \perp\!\!\!\perp B \mid S;\ A \perp\!\!\!\perp B \mid C, S\}$

(iii) $\{A \not\perp\!\!\!\perp C \mid B, S;\ B \not\perp\!\!\!\perp C \mid A, S;\ A \perp\!\!\!\perp B \mid S;\ A \perp\!\!\!\perp B \mid C, S\}$

For example, pattern (i) can be applied in Example 5.2 to find core c_2 by substituting Y for A, Q for B, X for C, and $\{W\}$ for S, thus eliminating the SAT solver call required to identify the core. Proofs of correctness and minimality for cores (i)–(iii) as well as four-variable patterns are listed in the online appendix for Paper III [83].

Incremental core extraction On any iteration, rather than immediately trying to satisfy D and all constraints $K \setminus H$, Dseptor incrementally adds constraints one by one until unsatisfiability. This alone has several benefits. The satisfiable formulas encountered before unsatisfiability yield upper bounds which are valuable for the bounds-based hardening discussed in the next paragraph. Considering a smaller set of constraints at each SAT solver call will naturally lead to smaller cores which reduces time spent on core minimization. A randomized order for adding constraints diversifies the found cores, which benefits the hitting set computations. Similar ideas have been explored for MaxSAT in e.g. the stratified approach of the WPM1 solver [9]. Dseptor implements a domain-specific refinement to incremental core extraction in the form of a dynamic partial encoding. Although the algorithm only tries to satisfy constraints not in H at each iteration, the clauses of the CNF encoding of D related to constraints in H remain active. The dynamic partial encoding explicitly disables the hard clauses related to the constraints which the current iteration does not try to satisfy.

Bounds-based hardening Motivated by simultaneous work on reduced-cost fixing for MaxSAT [16, 17] we used a type of lookahead procedure similar to the variable fixing and probing techniques in IP solving [140] to harden soft (in)dependence constraints during search. Given an upper bound U from a previously found feasible solution τ and a constraint k which is satisfied in that solution, we can solve the minimum-cost hitting set problem assuming k is violated to give a conditional lower bound $L_{\neg k}$. If $L_{\neg k} > U$, the soft constraint of k can be hardened since any solution which does not satisfy k has a cost higher than τ. A looser but computationally less costly conditional lower bound can be obtained by instead solving the LP relaxation of the hitting set problem. A domain-specific extension of this technique conditions the lower bound not on a single independence constraint but on the existence of an edge in the causal graph. If any edge e between nodes X and Y is present in the graph, *all* independence constraints of the form $X \perp\!\!\!\perp Y \mid S$ are falsified. A conditional lower bound L_e can then be computed by solving the hitting set IP (or LP relaxation) assuming that all constraints $X \perp\!\!\!\perp Y \mid S$ are violated. If

$L_e > U$, the problem can be simplified by enforcing the absence of e in the CNF encoding.

5.5 Experimental Evaluation

We experimentally evaluated the Dseptor solver on both synthetic and real-world data. Dseptor was compared against leading solvers for MaxSAT (OpenWBO [111], MaxHS [43], LMHS [136], Maxino [7], MSCG [115], WPM3 [10] and QMaxSAT [98]), ASP (Clingo [64]), and IP (CPLEX [84]). An overview of the experiments is given in Figures 5.3 and 5.4. In both cases Dseptor shows state-of-the-art performance, solving significantly more instances than other methods.

Figure 5.3 additionally shows the effect of each search technique introduced in Paper III. Each technique was individually disabled from Dseptor for the tests. The domain-specific cores (w.o. cores), incremental core extraction (w.o. incr), and bounds-based hardening (w.o. harden) each clearly contribute to the performance of the solver. Domain-specific cores individually give the largest benefit in terms of instances solved.

As seen with other implicit hitting set algorithms, Dseptor also has good anytime performance. The algorithm finds feasible solutions during search and, as shown in Paper III, often finds an optimal solution very fast, using most of the search time in proving optimality. After Dseptor, further progress was made in a custom branch-and-bound solver [130] which used many of the techniques introduced for this context in Paper III.

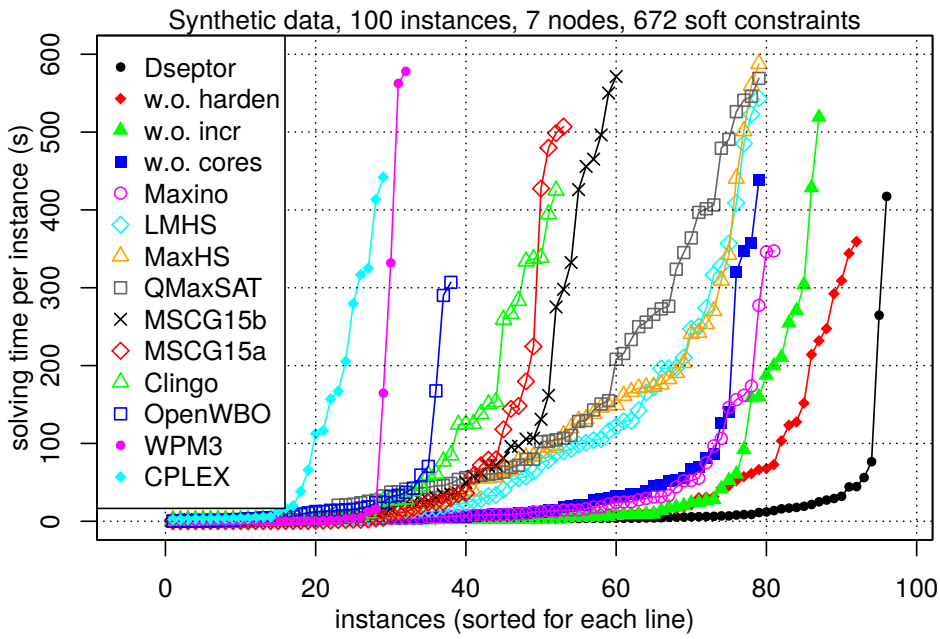


Figure 5.3: Evaluating the performance of Dseptor and the impact of search techniques.

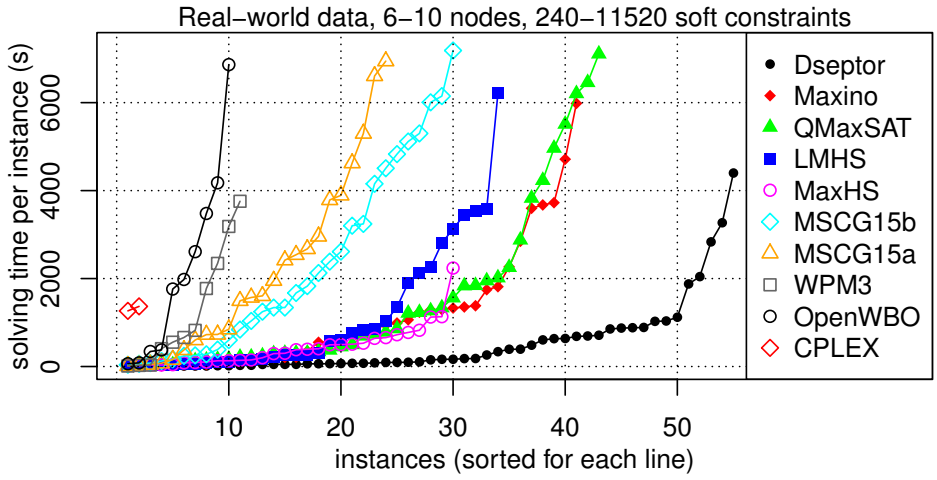


Figure 5.4: Performance of Dseptor with longer timeouts on larger, real-world datasets.

Chapter 6

Abductive Reasoning

In this chapter we discuss the contribution of this thesis to optimization in abductive reasoning, summarizing the work in Paper IV. Abductive reasoning, or abduction, is also known as inference to the best explanation [76]. Rather than deducing the logical consequence of some facts, abduction is a kind of reasoning used to formulate an explanation, such that the facts are a logical consequence of the explanation. In general many (possibly contradicting) explanations may exist, and abductive procedures do not claim to necessarily identify the true cause. Rather, the aim is to find e.g. the simplest or most probable explanation [50].

The study of abductive reasoning is a fairly modern development, having been formalized in the work of Peirce [61, 124] in the early 20th century. There are various formalisms for abduction, including probabilistic abduction, set-cover abduction, and logic-based abduction [125]. We consider abduction problems in a propositional logic setting. Abductive reasoning problems are encountered in a number of contexts including diagnosis [59, 125], machine learning [87, 156], planning [60], law [143], medicine [103], and natural language processing [79, 119].

Many knowledge representation and reasoning problems, including abduction, are known to belong to complexity classes beyond NP [53, 55, 58, 148]. More specifically the propositional abduction problem we consider is Σ_2^P-hard [56]. In instantiating the implicit hitting set framework for this task in Paper IV, we show the robustness of the approach for solving optimization problems beyond NP in the polynomial hierarchy.

We begin with a formal definition of the propositional abduction problem and related concepts. We then present an instantiation of the generic hitting set algorithm for propositional abduction. This instantiation uses separate entailment and consistency checks in the core extractor to tackle the Σ_2^P-hard optimization problem. We discuss our implementation of the

AbHS algorithm as well as a refinement (AbHS+) which gives substantial performance gains in practice. The chapter concludes with an overview of an experimental evaluation of AbHS and a discussion of recent related work on solving abduction problems and application of abduction solvers.

6.1 Propositional Abduction Problems

Formally, a propositional abduction problem instance $P = (V, H, M, T, c)$ is a tuple of sets of variables V, hypotheses H, manifestations (or observations) M, a theory T, and a cost function $c : H \to \mathbb{N}$. Here V is a (finite) set of Boolean variables, H and M are sets of propositional formulas, and T is a propositional formula. We extend the domain of the cost function to subsets of H by simply summing the costs of their elements $c(S) = \sum_{s \in S} c(s)$.

Similarly to our treatment of CNF formulas (in Chapter 2) we use H (and $S \subseteq H$) to also refer to the conjunction of their elements. A subset of hypotheses $S \subseteq H$ is an *explanation* of P if $T \wedge S$ is both consistent and entails M (denoted by $T \wedge S \models M$). This entailment holds if $T \wedge S \wedge \neg M$ is inconsistent (unsatisfiable). We denote the set of all explanations of P with

$$Expl(P) = \{S \subseteq H \mid T \wedge S \not\models \bot, T \wedge S \models M\}$$

and the set of minimum-cost explanations (with regard to c) with

$$Expl_c(P) = \operatorname*{arg\,min}_{E \in Expl(P)} c(E).$$

The problem of checking whether an explanation exists is Σ_2^P-complete (alternatively $\mathrm{NP}^{\mathrm{NP}}$) for propositional theories [56], i.e., it can be decided in nondeterministic polynomial time with access to an NP oracle. This is in contrast to Chapter 4 where the decision problem (for MaxSAT) is NP-complete.

A core of an abduction instance P is a set of hypotheses which is not a subset of any explanation. That is, $C \subseteq H$ is a core if there does not exist any explanation S for which $C \subseteq S \subseteq H$. Unlike MaxSAT cores which are unsatisfiable, an abduction core C can be consistent with the theory T.

6.2 Instantiating IHS for Abduction

In instantiating the implicit hitting set for propositional abduction we make, without loss of generality, some assumptions on the form of the

input. In the following, the theory T is a propositional formula in CNF form and M and H are sets of literals (unit clauses). Given this restriction, manifestations and hypotheses comprised of multiple clauses can be represented by e.g. a group encoding [77].

Algorithm 2 presents AbHS, our instantiation of the implicit hitting set algorithm for propositional abduction. AbHS first initializes an empty set of cores K and an empty explanation candidate S. An initial SAT solver call (Line 3) then checks that the entire set of hypotheses entails M, $T \wedge H \models M$, or in other words checks that $T \wedge H \wedge \neg M$ is unsatisfiable. The procedure then enters the main implicit hitting set loop, which consists of two successive SAT solver invocations. The first (Line 6) ensures that the candidate solution S entails the manifestations. The second (Line 10) checks that S is consistent with the theory T. If either check is unsuccessful, an appropriate core is added to K. If the first fails ($T \wedge S \not\models M$), a core can be formed from the subset of hypotheses not satisfied by the truth assignment τ. This is to say that a hypothesis not active in the violating assignment τ must be enforced to entail M. If the second fails ($T \wedge S$ is not satisfiable), then since S is a minimal hitting set for K we add to K a core requiring at least one hypothesis in $H \setminus S$. Before every iteration, S is recomputed to be the minimum-cost hitting set of the set of cores K. If S grows to the entire set of hypotheses H, we know from the initial check of Line 3 that there is no solution. If both the entailment and consistency check are successful on any iteration of the algorithm, then S is a minimum-cost explanation and the algorithm terminates. Figure 6.1 relates the structure of this algorithm to the implicit hitting set framework.

The AbHS+ variant also introduced in Paper IV departs somewhat from the pure implicit hitting set framework. We observed that for small S, the core $H \setminus S$ added on Line 14 can be very large. This core is expressed as an IP constraint given a Boolean variable x_i for each hypothesis $h_i \in H$ as

$$\sum_{h_i \in (H \setminus S)} x_i \geq 1.$$

Given that S is by construction minimal (and thus no subset of it will be considered on subsequent iterations), a more concise way of expressing the same condition is the constraint

$$\sum_{h_i \in S} x_i < |S|$$

to indicate instead that at least one hypothesis in S cannot be included in any explanation. As noted in [85], the inclusion of these constraints may

Algorithm 2 An implicit hitting set algorithm for abduction.

```
1: procedure AbHS(V, H, M, T, c)
2:     K ← ∅; S ← ∅
3:     (sat, τ) ← SAT(T ∧ H ∧ ¬M)
4:     if sat then return "no solution"
5:     while S ≠ H do
6:         (sat, τ) ← SAT(T ∧ S ∧ ¬M)
7:         if sat then
8:             K ← K ∪ {{h ∈ H | τ(h) = 0}}                ▷ T ∧ S ⊭ M
9:         else
10:            (sat, τ) ← SAT(T ∧ S)
11:            if sat then
12:                return S
13:            else
14:                K ← K ∪ {{H \ S}}                       ▷ T ∧ S ⊨ ⊥
15:        S ← MCHS(K, c)
16:    return "no solution"
```

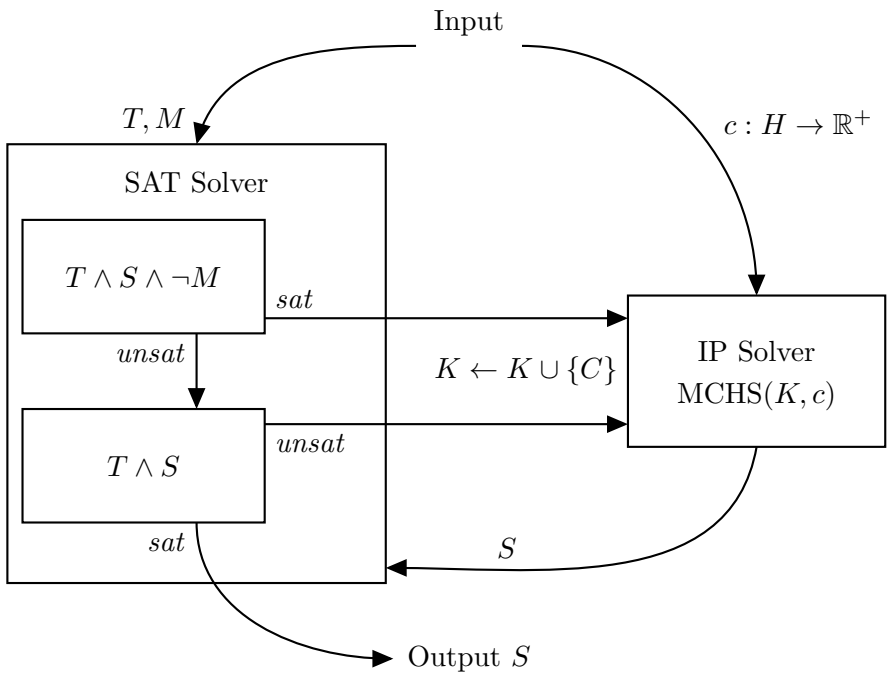


Figure 6.1: The structure of Algorithm 2.

cause the IP to have no solutions and the loop of Algorithm 2 should be modified to terminate in this case.

As an alternative to the implementation of AbHS using two SAT solver calls per iteration, the IHS algorithm could be instantiated with a Σ_2^P-oracle such as a quantified Boolean formula (QBF) solver. However, this approach was found to perform poorly in practice with current QBF solvers [85].

6.3 Experimental Evaluation

In the absence of a standard benchmark set for propositional abduction problems, a set of instances was generated from MaxSAT evaluation 2014 benchmark instances [11]. An abduction instance (V, H, M, T, c) was constructed from a MaxSAT instance (F_h, F_s, w) as follows. Soft clauses F_s were used as hypotheses H, and hard clauses F_h as the theory T. From an optimal solution to the MaxSAT instance, which satisfies soft clauses $F'_s \subset F_s$, a subset of literals entailed by $F'_s \wedge F_h$ is used as manifestations M. Random subsets of 5, 10, and 15 literals were selected for each of the 547 instances, giving a total of 1641 abduction instances.

A prototype implementation of the AbHS algorithm (and AbHS+ variant) was tested against a disjunctive ASP encoding. An abduction instance is encoded in ASP by a standard guess-and-check technique in which consistency is checked with simple ASP constraints. Checking entailment is more involved, using a so-called saturation technique with disjunctive rules [57]. For details, the ASP encoding is available on the AbHS solver website [138]. We used Clingo 4.5.3 [64] as the ASP solver. The core-guided algorithm of Clingo was also tested, but it solved fewer instances than the default branch-and-bound algorithm for all values of $|M|$. Results of the comparison are summarized in Figure 6.2.

Each algorithm was run with a per-instance timeout of 1800 seconds. Instances with a larger number of manifestations were clearly harder for each algorithm. While the ASP approach solved more instances than AbHS with $|M| = 10$ and $|M| = 15$, the AbHS+ variant clearly dominated both in each case.

6.4 Related Work

Prior work on propositional abduction problems suggested similar approaches which are also based on computing minimal hitting sets [139]. However, these algorithms focus on enumeration of all explanations, and

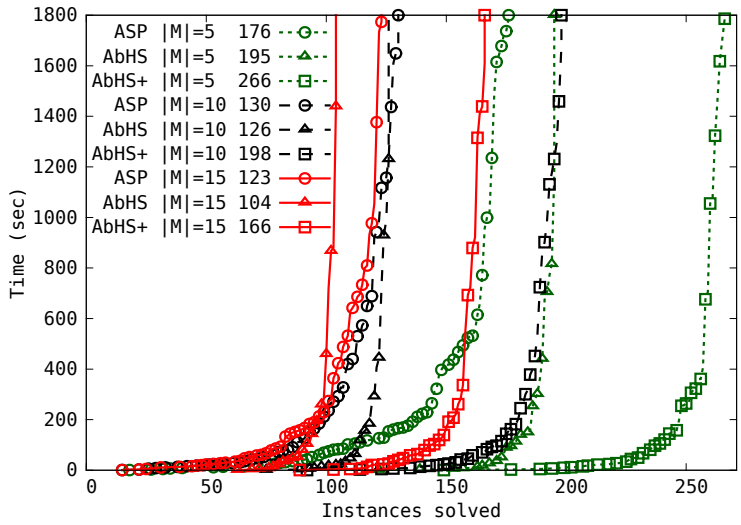


Figure 6.2: Comparison of propositional abduction solvers grouped by size of manifestation set M.

the implementations have not been empirically tested to the best of our knowledge.

The implicit hitting set approach for abduction was further developed in [85], where the check for consistency is integrated into the hitting set solver. This was done by substituting the IP solver of AbHS with a MaxSAT solver and adding the relevant theory constraints. As explained in [85], this has the potential to result in an exponential reduction in the number of SAT solver calls required. In practice the new solver was shown to have better performance on unweighted instances in terms of number of instances solved.

Recently, this approach was applied in [87] for computing minimal explanations for neural networks. The task of determining an explanation for the predicted label of a machine learning model for a given input can be seen as an abductive reasoning task. Specifically, in [87] a propositional abduction procedure is used to search for a cardinality-minimal set of features of an input, which together with the parameters and structure of the neural network entail the predicted label.

Chapter 7

Answer Set Optimization

This chapter summarizes the contributions of this thesis to optimization in answer set programming. Answer set programming (ASP) [70, 104, 118] is a relatively recent declarative programming paradigm. It borrows much of the syntax of traditional logic programming (e.g. Prolog [146]) but differs in its use of the stable model semantics [69] and negation as failure [39]. Today, various solvers for ASP are available. Examples include smodels [144], DLV [100], Wasp [6], and clasp [64]. Most early optimization procedures for ASP were branch-and-bound algorithms but more recently core-guided solvers [6, 8] have been developed. These are based on algorithms similar to those used in MaxSAT [114]. Optimization with answer set programming sees a variety of applications including solving problems in robotics [141], vehicle routing [67], and scheduling [1, 5, 20]. In Paper V we develop and evaluate ASP-HS, a new answer set optimization procedure based on the implicit hitting set paradigm.

We begin with the necessary definitions of answer set programming and optimization. Our focus is on ground (variable-free) programs with disjunctive rules. We discuss stable models, Clark's completion, and optimization over soft literals in this context. We then introduce our instantiation of the implicit hitting set algorithm for answer set programming. The main contribution of the chapter, the description and evaluation of various search techniques developed for ASP-HS, follows. In particular, we make an interesting concrete observation on the impact of weight precision on solving times.

7.1 Answer Set Programs

Answer set programs are written using an extended logic programming notation. Programs consist of atoms and inference rules. Atoms represent a claim about the world which may be true or false. Inference rules encode the relationships between these claims. A solution is a set of true atoms called a stable model [144].

The ASP language specification allows for variables and various types of syntactic shortcuts in the form of choice rules and aggregate relations (including cardinality constraints) [33]. These aggregates can be expressed in terms of simpler rules. Variables are often completely grounded (instantiated) away before solving [68,150] (although so-called lazy grounding schemes also exist [99,120]).

We target ground programs over a set of propositional atoms $\mathcal{A}$. A literal l of an atom $p \in \mathcal{A}$ can be positive p or negative $\sim p$. Here $\sim$ denotes default negation or negation-as-failure [39], not classical negation $\neg$. An interpretation I (analogous to a SAT model) is a subset of $\mathcal{A}$. Atoms in I are true and atoms in $\mathcal{A} \setminus I$ are false. For our purposes an answer set program Π is a set of rules of the form

$$p_1 \vee \cdots \vee p_m \leftarrow l_1 \wedge \cdots \wedge l_n$$

where $p_1, \ldots, p_m$ are atoms and $l_1, \ldots, l_n$ literals of $\mathcal{A}$. If $m > 1$ the rule is *disjunctive*. The inclusion of disjunctive rules pushes the complexity of the decision problem of answer set programming to the second level of the polynomial hierarchy [54]. For a rule r the atoms $p_1, \ldots, p_m$ are the *head* of the rule, $H(r)$, and the literals $l_1, \ldots, l_n$ are the *body* $B(r)$. We use $B(r)^+$ and $B(r)^-$ to denote the atoms occurring in the positive and negative literals of the body.

The semantics of these rules are such that if the conditions of the body hold, then at least one of the atoms in the head must be included in a solution. More formally I is a model of a rule r, $I \models r$, if $H(r) \cap I \neq \emptyset$ whenever $B(r)^+ \subseteq I$ and $B(r)^- \cap I = \emptyset$. I is a model of a program Π, $I \models \Pi$, if I is a model of every rule $r \in \Pi$. A rule with an empty body

$$p_1 \vee \cdots \vee p_m \leftarrow$$

is commonly known as a *fact*, which is essentially a clause $(p_1 \vee \cdots \vee p_m)$. A rule with an empty head

$$\leftarrow l_1 \wedge \cdots \wedge l_n$$

is an *integrity constraint* which corresponds to a clause $(\neg l_1 \vee \cdots \vee \neg l_n)$, as the empty head cannot be satisfied.

Stable models (or answer sets) [69] formalize the notion that the inclusion of every element in the solution must be justified by the rules of the program. They are defined using a *reduct* Π^I of the program Π with respect to an interpretation I. The reduct Π^I is obtained by

1. first removing each rule r such that $B(r)^- \cap I \neq \emptyset$,
2. then removing negative literals in the remaining rule bodies.

Intuitively, we can see Π^I as the set of potentially applicable inference rules given I. Step 1 removes from Π inference rules that cannot be applied because they contain the negation of an atom included in I. Step 2 then removes the remaining negative literals because they are trivially satisfied due to not being included in I. An interpretation I is a stable model of Π if $I \models \Pi$ and there is no $J \subset I$ such that $J \models \Pi^I$. A program Π is coherent if it has a stable model and incoherent otherwise.

Example 7.1. Consider a disjunctive answer set program Π similar to the example of Paper V with rules

$$a \vee c \leftarrow {\sim}b \wedge {\sim}d, \quad a \leftarrow c \wedge {\sim}d, \quad b \vee d \leftarrow {\sim}c, \quad c \leftarrow a \wedge {\sim}b,$$
$$\leftarrow s_a \wedge {\sim}a, \quad \leftarrow s_b \wedge {\sim}b, \quad \leftarrow s_c \wedge {\sim}c, \quad \leftarrow s_d \wedge {\sim}d,$$
$$s_a \vee s_b \vee s_c \vee s_d \leftarrow .$$

The program Π has stable models $I_1 = \{b, s_b\}$, $I_2 = \{d, s_d\}$, $I_3 = \{c, a, s_a\}$, and $I_4 = \{c, a, s_c\}$. The reduct Π^{I_1} is

$$a \leftarrow c, \quad b \vee d \leftarrow , \quad \leftarrow s_a, \quad \leftarrow s_c, \quad \leftarrow s_d,$$
$$s_a \vee s_b \vee s_c \vee s_d \leftarrow$$

and one can check that no proper subset of I_1 is a model of Π^{I_1}, hence I_1 is indeed a stable model.

We use the Clark's completion [39] as a tool for reasoning about answer set programs. In the non-disjunctive case the completion of a program Π is a propositional logic formula in which we interpret rules r of Π as implications $B(r) \rightarrow H(r)$ and add

$$p \rightarrow \bigvee_{r : p \in H(r)} B(r)$$

for each atom $p \in \mathcal{A}$. These expressions state that if p is true, then the body of a rule r with p as its head is satisfied. The solutions of the completion correspond to supported models of Π, a superset of the stable models of

Π. Extensions of the Clark's completion to disjunctive programs have also been formulated [4].

7.2 Instantiating IHS for ASP

A program Π may have many stable models. We denote by $SM(\Pi)$ the set of all stable models of Π. We define optimization over these stable models in terms of *soft atoms*. Alternatively we could optimize over weak constraints, similarly to how optimization for MaxSAT could be defined either over labels or over soft clauses in Chapter 4. Let $\mathcal{W}$ be a finite set of soft atoms and let $weight : \mathcal{W} \to \mathbb{N}$ assign a positive integer weight $weight(p)$ to each soft atom. The cost of an interpretation I is the weight of soft atoms not included in it,

$$\mathcal{W}(I) = \sum_{p \in \mathcal{W} \setminus I} weight(p).$$

An interpretation $\hat{I}$ is an optimal stable model if

$$\hat{I} \in \arg\min_{I \in SM(\Pi)}(\mathcal{W}(I)).$$

Example 7.2. Continuing Example 7.1, suppose that the set of soft atoms $\mathcal{W}$ is $\{s_a, s_b, s_c, s_d\}$ and *weight* assigns weights $\{1, 2, 4, 8\}$, respectively, to the soft atoms. I_2 is an optimal stable model of Π as the stable models I_1, I_2, I_3, and I_4 have cost 13, 7, 14, and 11, respectively.

The development of core-guided solvers for ASP [6, 8] opened the door for the implementation of an implicit hitting set based optimization procedure. Cores of answer set programs are defined in a similar manner to MaxSAT cores in Chapter 4. A core of a program Π is a subset of soft atoms $\kappa \subseteq \mathcal{W}$ that cannot be simultaneously included in any stable model of Π. That is, if $\kappa \subseteq I$ then $I \notin SM(\Pi)$.

Example 7.3. Continuing Example 7.2, Π has no stable models with more than one soft atom of $\mathcal{W}$ so any pair of atoms of $\mathcal{W}$, e.g. $\{s_a, s_b\}$, is a core.

Using a core-guided ASP solver as an unsatisfiable core extractor, it is then straightforward to instantiate the IHS framework for ASP optimization. This allows us to find a core with respect to a set of soft literals as assumptions. When solving Π w.r.t. a set S, we will either find $I \in SM(\Pi)$

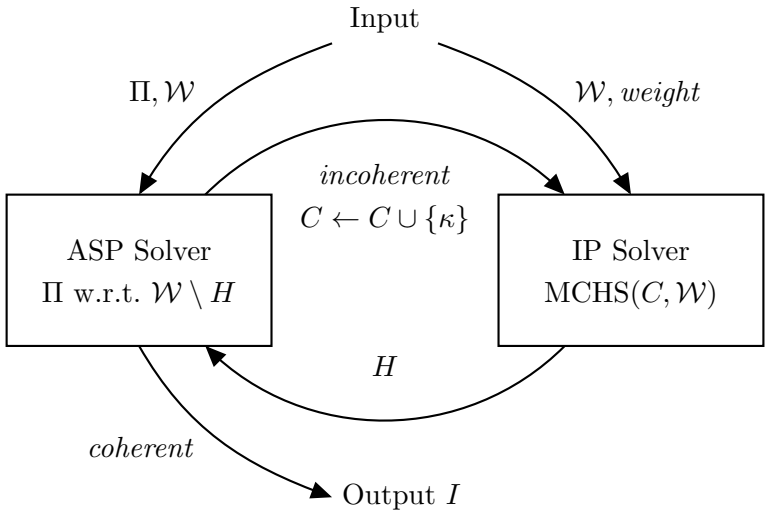


Figure 7.1: The implicit hitting set algorithm for ASP.

such that $S \subseteq I$ or a core $\kappa \subseteq S$ if no such stable model exists. Figure 7.1 shows the familiar implicit hitting set loop in the context of ASP. That this process yields an optimal stable model follows from Theorem 3.1.

7.3 Search Techniques

We implemented the implicit hitting set algorithm for ASP in our solver, ASP-HS. The solver incorporates many search techniques on top of the implicit hitting set framework. Some of these are known from implicit hitting set algorithms in other domains, while others are novel for the ASP instantiation. The former category includes core minimization, disjoint core computations, and reduced cost fixing while the latter consists of techniques exploiting the Clark's completion of the program.

Core minimization We use a simple destructive algorithm for core minimization which was shown to be effective in MaxHS [45] and LMHS [135]. In short, a subset-minimal core is computed by iteratively discarding unnecessary elements from a found core κ. We imposed resource constraints on the minimization process which prevent it from consuming more than a fixed fraction of the total search time.

Disjoint cores Disjoint cores are used in the same manner as in the LMHS MaxSAT solver [135]. With the purpose of increasing core diversity, ASP-HS computes a disjoint set of cores at each iteration of the implicit hitting set loop. The disjoint core computation finds a feasible solution at each iteration. These feasible solutions give upper bounds which can be exploited by reduced cost fixing.

Reduced cost fixing The IP solving technique of reduced-cost fixing is implemented following its recent application for MaxSAT [16, 17]. At each iteration, an LP relaxation of the problem is solved to yield reduced-cost values for the optimization variables. On an intuitive level for ASP (with respect to an optimal solution to the LP relaxation), the reduced cost of an atom p not in the answer set represents the amount by which the cost of an optimal solution would increase if we required p to be in an answer set. Similarly to the bounds-based hardening in Section 5.4, this gives us a conditional lower bound for answer sets including p, which can make it possible to simplify the problem by fixing the value of its respective soft literal during search.

Clark's completion We developed methods that leverage the Clark's completion specifically for ASP-HS. The Clark's completion (recall Section 7.1) is a relaxation of the answer set program, and thus we could solve it as an initial step to obtain a lower bound on the cost of optimal solutions. However, solving the completion can be too time-consuming in practice and we solve an LP relaxation of the completion instead, as explained in more detail in Paper V. Computing an initial lower bound can benefit search by enabling reduced cost fixing or even cause the early termination of the algorithm if it matches an upper bound obtained from minimization or disjoint cores.

Non-core constraints We can identify rules consisting only of soft literals in the answer set program and pass these to the optimizer as additional information. Our motivation is similar to the use of non-core constraints in MaxHS for MaxSAT [45]. Unlike the case of MaxSAT, we look for these constraints in the Clark's completion of the answer set program. If any constraint in the completion of Π contains only atoms in $\mathcal{W}$, it can be used by the IP solver to refine the hitting sets produced and thus potentially reduce the number of iterations required for the IHS algorithm to terminate.

7.4 Experimental Evaluation

An experimental evaluation of ASP-HS and other solvers for ASP optimization was performed in Paper V. The Wasp 2.0 ASP solver [6] was used as the unsatisfiable core extractor and IBM CPLEX 12.7 [84] as the optimizer. Wasp supports core extraction for disjunctive programs [3], extending the reach of ASP-HS to Σ_2^P-hard [54,57] optimization problems. We gathered a benchmark set of ASP optimization instances from the 2015 and 2017 ASP Competitions [65,66], the Asparagus instance repository [89], and instances of a causal structure discovery problem [82]. To provide a more balanced comparison, 20 instances were chosen for each instance family as in the ASP Competitions. For instance families not from the ASP Competitions, a set of 20 instances was chosen randomly. In total 16 different instance families (11 weighted and 5 unweighted) were considered, for a total of 320 instances.

Figure 7.2 shows the impact of individual search techniques on the performance of ASP-HS on (W)eighted and (U)nweighted instances. Each search technique considered in Section 7.3 is disabled one at a time (e.g. "No disjoint") to show its contribution to the number of instances solved by ASP-HS. The largest contributions to the number of instances solved are observed on weighted instances, where core minimization and disjoint cores have the greatest impact. The ASP-specific non-core constraints and Clark's completion lower bounds have a smaller, but noticeable, impact for both unweighted and weighted instances.

Figure 7.3 compares ASP-HS to Wasp 2.0 [6] and the core-guided (usc) and branch-and-bound (bb) algorithms of Clasp 3.3.2 [8,64]. Compared to the core-guided approaches of Wasp and Clasp-usc, ASP-HS is very competitive on weighted instances, while on unweighted instances it outperforms the branch-and-bound approach of Clasp-bb. A more detailed summary of results by instance family is given in Paper V.

The strength of the implicit hitting set approach is seen in particular in the weighted instances. A more in-depth look at the impact of weight precision is given by Table 7.1 where we take the Bayes instance family as an example. By chance, we noticed that the weight precision of these instances had been artificially reduced by dividing all weights by 1000. This gave a natural way of investigating the impact of weight precision. We varied the value of the divisor from 1000 to 1, to create instance families Bayes1000, Bayes100, Bayes10, and Bayes1. That is, Bayes1 is the instance set with maximum available precision and Bayes1000 is the unmodified instance set. We see a sharp decline in the performance of Wasp as precision increases while the performance of ASP-HS is fairly consistent.

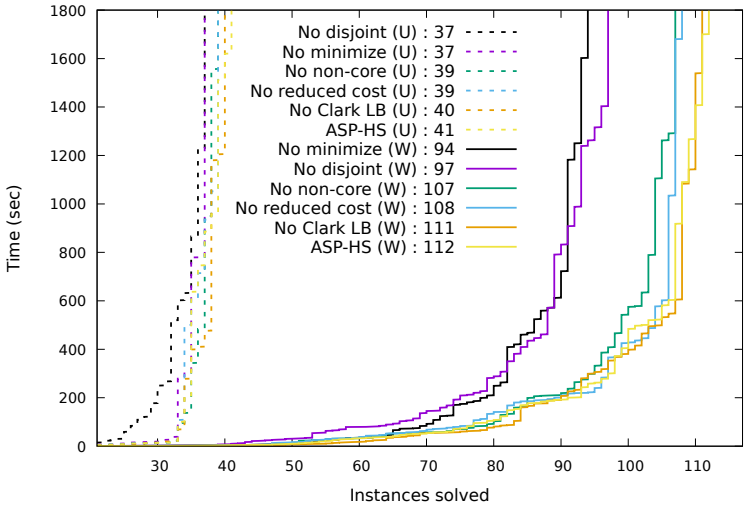


Figure 7.2: Impact of specific search techniques on ASP-HS.

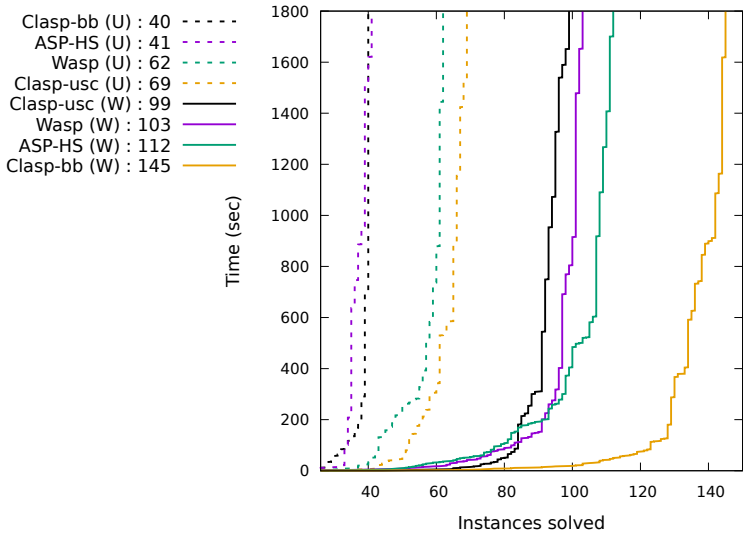


Figure 7.3: Comparison of ASP-HS to other state-of-the-art solvers.

	ASP-HS		Wasp	
Family	solved	time	solved	time
Bayes1 (60)	**23**	73881.28	5	99335.61
Bayes10 (60)	**23**	73273.87	6	97480.60
Bayes100 (60)	**23**	72843.50	12	89454.11
Bayes1000 (60)	27	64406.41	**35**	51484.31
Total	**96**	284405.06	58	337754.63

Table 7.1: Impact of weight precision on the Bayes instance family.

Chapter 8

Conclusion

This thesis contributes practical applications of implicit hitting set algorithms to a number of constraint optimization settings. We instantiated the implicit hitting set algorithm for MaxSAT (through labeled CNF formulas), causal structure discovery, propositional abduction, and answer set optimization. We released an open-source implementation of each algorithm and performed empirical evaluations of each implementation to show that they improve or complement the state of the art.

Further work on implicit hitting set algorithms will benefit from deeper insight into how to compute cores and minimum-cost hitting sets for the task. Many applications of the implicit hitting set algorithm would benefit from developing the incremental use of SAT solvers for core extraction, such as recent work on using assumptions [78]. Similarly further development of core extraction procedures for e.g. quantified satisfiability would likely improve implicit hitting set algorithms for problems further up the polynomial hierarchy. Implementations of implicit hitting set algorithms commonly use IP or MaxSAT solvers to tackle the sequences of minimum-cost hitting set problems. Their efficient iterative or incremental use is an important consideration. While the choice of core extractor is often dictated by the application, there is more freedom in how to solve the hitting set problems. Algorithms specifically targeted towards incrementally solving instances of minimum-cost hitting set problems have the potential to benefit nearly all instantiations of implicit hitting set algorithms.

We expect the implicit hitting set approach to give more opportunities for parallelization than e.g. purely SAT-based algorithms, as SAT solvers have proven challenging to parallelize [19,75]. In addition to parallelization of the IP solver, cores can be extracted in a parallel or distributed manner. Due to the sequential nature of the problem it remains a challenge to efficiently put together the results of parallel computations.

Following the application of reduced-cost fixing techniques [16, 17] to implicit hitting set algorithms, and given the wealth of literature on solving IP problems, it seems natural to ask what other IP techniques could be successfully applied to implicit hitting set algorithms. The importance of core diversity for implicit hitting set algorithms has long been noted, but has yet to be formally characterized and justified. This could yield insights into how to guide the search for cores.

The success of Dseptor in specializing the implicit hitting set approach to a specific problem domain suggests that a similar approach could be worthwhile to investigate for other problems as well. Similar ideas of IHS-specific encodings and domain-specific cores might likewise advance the state of the art for other problems where efficient core extractors exist.

References

[1] M. Abseher, M. Gebser, N. Musliu, T. Schaub, and S. Woltran. Shift design with answer set programming. In F. Calimeri, G. Ianni, and M. Truszczynski, editors, *Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 9345 of *Lecture Notes in Computer Science*, pages 32–39. Springer, 2015.

[2] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, Jul 2009.

[3] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, and F. Ricca. Evaluation of disjunctive programs in WASP. In M. Balduccini, Y. Lierler, and S. Woltran, editors, *Proceedings of the 15th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 11481 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2019.

[4] M. Alviano and C. Dodaro. Completion of disjunctive logic programs. In S. Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 886–892. IJCAI/AAAI Press, 2016.

[5] M. Alviano, C. Dodaro, and M. Maratea. An advanced answer set programming encoding for nurse scheduling. In F. Esposito, R. Basili, S. Ferilli, and F. A. Lisi, editors, *Proceedings of the XVIth International Conference of the Italian Association for Artificial Intelligence*, volume 10640 of *Lecture Notes in Computer Science*, pages 468–482. Springer, 2017.

[6] M. Alviano, C. Dodaro, J. Marques-Silva, and F. Ricca. Optimum stable model search: algorithms and implementation. *Journal of Logic and Computation*, 2015.

[7] M. Alviano, C. Dodaro, and F. Ricca. A MaxSAT algorithm using cardinality constraints of bounded size. In Q. Yang and M. J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 2677–2683. AAAI Press, 2015.

[8] B. Andres, B. Kaufmann, O. Matheis, and T. Schaub. Unsatisfiability-based optimization in clasp. In A. Dovier and V. S. Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2012.

[9] C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy. Improving SAT-based weighted MaxSAT solvers. In M. Milano, editor, *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming*, volume 7514 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2012.

[10] C. Ansótegui and J. Gabàs. WPM3: an (in)complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017.

[11] J. Argelich, C. M. Li, F. Manyà, and J. Planes. Ninth Max-SAT evaluation. `http://www.maxsat.udl.cat/14/benchmarks/`, 2014. Accessed: 2019-09-05.

[12] J. Argelich, C. M. Li, F. Manyà, and J. Planes. Tenth Max-SAT evaluation. `http://www.maxsat.udl.cat/15/results/`, 2015. Accessed: 2019-09-05.

[13] M. F. Arif, C. Mencía, and J. Marques-Silva. Efficient MUS enumeration of Horn formulae with applications to axiom pinpointing. In M. Heule and S. A. Weaver, editors, *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing*, volume 9340 of *Lecture Notes in Computer Science*, pages 324–342. Springer, 2015.

[14] G. Audemard, J. Lagniez, and L. Simon. Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In M. Järvisalo and A. Van Gelder, editors, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *Lecture Notes in Computer Science*, pages 309–317. Springer, 2013.

[15] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In C. Boutilier, editor, *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 399–404. ijcai.org, 2009.

[16] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko. Reduced cost fixing in MaxSAT. In J. C. Beck, editor, *Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017.

[17] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko. Reduced cost fixing for maximum satisfiability. In J. Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 5209–5213. ijcai.org, 2018.

[18] J. Bailey and P. J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In M. V. Hermenegildo and D. Cabeza, editors, *Proceedings of the 7th International Symposium on Practical Aspects of Declarative Languages*, volume 3350 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2005.

[19] T. Balyo and C. Sinz. Parallel satisfiability. In *Handbook of Parallel Constraint Reasoning*, chapter 1, pages 3–25. Springer, 2018.

[20] M. Banbara, K. Inoue, B. Kaufmann, T. Okimoto, T. Schaub, T. Soh, N. Tamura, and P. Wanko. *teaspoon*: solving the curriculum-based course timetabling problems with answer set programming. *Annals of Operations Research*, 275(1):3–37, 2019.

[21] M. Bartlett and J. Cussens. Integer linear programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017.

[22] A. Belov and J. Marques-Silva. Accelerating MUS extraction with recursive model rotation. In P. Bjesse and A. Slobodová, editors, *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, pages 37–40. FMCAD Inc., 2011.

[23] A. Belov and J. Marques-Silva. Generalizing redundancy in propositional logic: Foundations and hitting sets duality. *CoRR*, abs/1207.1257, 2012.

[24] A. Belov, A. Morgado, and J. Marques-Silva. SAT-based preprocessing for MaxSAT. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 8312 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2013.

[25] J. Berg, A. Hyttinen, and M. Järvisalo. Applications of MaxSAT in data analysis. In D. L. Berre and M. Järvisalo, editors, *Proceedings of Pragmatics of SAT 2015 and 2018*, volume 59 of *EPiC Series in Computing*, pages 50–64. EasyChair, 2018.

[26] J. Berg and M. Järvisalo. Impact of SAT-based preprocessing on core-guided MaxSAT solving. In M. Rueher, editor, *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming*, volume 9892 of *Lecture Notes in Computer Science*, pages 66–85. Springer, 2016.

[27] J. Berg and M. Järvisalo. Unifying reasoning and core-guided search for maximum satisfiability. In F. Calimeri, N. Leone, and M. Manna, editors, *Proceedings of the 16th European Conference on Logics in Artificial Intelligence*, volume 11468 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2019.

[28] J. Berg, P. Saikko, and M. Järvisalo. Re-using auxiliary variables for MaxSAT preprocessing. In *Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence*, pages 813–820. IEEE Computer Society, 2015.

[29] J. Berg, P. Saikko, and M. Järvisalo. Subsumed label elimination for maximum satisfiability. In G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, editors, *Proceedings of the 22nd European Conference on Artificial Intelligence*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 630–638. IOS Press, 2016.

[30] A. Biere. Yet another local search solver and Lingeling and friends entering the SAT competition 2014. In A. Balint, A. Belov, M. Heule, and M. Järvisalo, editors, *Proceedings of SAT Competition 2014*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, pages 39–40. University of Helsinki, 2014.

[31] A. Boneh. Identification of redundancy by a set-covering equivalence. In J. P. Brans, editor, *Proceedings of the Tenth International Con-*

ference on Operational Research, volume 84, pages 407–422. Elsevier, 1984.

[32] K. Bunte, M. Järvisalo, J. Berg, P. Myllymäki, J. Peltonen, and S. Kaski. Optimal neighborhood preserving visualization by maximum satisfiability. In C. E. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1694–1700. AAAI Press, 2014.

[33] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, and T. Schaub. ASP-Core-2: Input language format. ASP Standardization Working Group, 2015.

[34] K. Chandrasekaran, R. M. Karp, E. Moreno-Centeno, and S. Vempala. Algorithms for implicit hitting set problems. In D. Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 614–629. SIAM, 2011.

[35] J. W. Chinneck and E. W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168, 1991.

[36] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[37] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. A modular approach to MaxSAT modulo theories. In M. Järvisalo and A. Van Gelder, editors, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *Lecture Notes in Computer Science*, pages 150–165. Springer, 2013.

[38] T. Claassen and T. Heskes. A Bayesian approach to constraint based causal inference. In N. de Freitas and K. P. Murphy, editors, *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 207–216. AUAI Press, 2012.

[39] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Symposium on Logic and Data Bases*, Advances in Data Base Theory, pages 293–322, New York, 1977. Plenum Press.

[40] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[41] H. Crowder, E. L. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[42] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2(4):393–410, 1954.

[43] J. Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. PhD thesis, University of Toronto, 2013.

[44] J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In J. H. Lee, editor, *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

[45] J. Davies and F. Bacchus. Exploiting the power of MIP solvers in MAXSAT. In M. Järvisalo and A. Van Gelder, editors, *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

[46] J. Davies and F. Bacchus. Postponing optimization to speed up MAXSAT solving. In C. Schulte, editor, *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013.

[47] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[48] E. Delisle and F. Bacchus. Solving weighted CSPs by successive relaxations. In C. Schulte, editor, *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 273–281. Springer, 2013.

[49] E. Demirovic, N. Musliu, and F. Winter. Modeling and solving staff scheduling with partial weighted maxSAT. *Annals of Operations Research*, 275(1):79–99, 2019.

[50] I. Douven. Abduction. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2017 edition, 2017.

[51] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In F. Bacchus and T. Walsh, editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

[52] N. Eén and N. Sörensson. An extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[53] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.

[54] T. Eiter and G. Gottlob. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In D. Miller, editor, *Proceedings of the 1993 International Symposium on Logic Programming*, pages 266–278. MIT Press, 1993.

[55] T. Eiter and G. Gottlob. Propositional circumscription and extended closed-world reasoning are Π_2^P-complete. *Theoretical Computer Science*, 114(2):231–245, 1993.

[56] T. Eiter and G. Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, 1995.

[57] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[58] T. Eiter and T. Lukasiewicz. Default reasoning from conditional knowledge bases: Complexity and tractable cases. *Artificial Intelligence*, 124(2):169–241, 2000.

[59] B. el Ayeb, P. Marquis, and M. Rusinowitch. Preferring diagnoses by abduction. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):792–808, 1993.

[60] K. Eshghi. Abductive planning with event calculus. In R. A. Kowalski and K. A. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 562–579. MIT Press, 1988.

[61] K. T. Fann. *Peirce's Theory of Abduction*. Springer Netherlands, 1970.

[62] K. Fazekas, F. Bacchus, and A. Biere. Implicit hitting set algorithms for maximum satisfiability modulo theories. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Proceedings of the 9th International Joint Conference on Automated Reasoning*, volume 10900 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2018.

[63] J. Franco and J. Martin. A history of satisfiability. In *Handbook of Satisfiability*, chapter 1, pages 3–55. IOS Press, 2009.

[64] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. *clasp*: A conflict-driven answer set solver. In C. Baral, G. Brewka, and J. S. Schlipf, editors, *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 4483 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 2007.

[65] M. Gebser, M. Maratea, and F. Ricca. The design of the sixth answer set programming competition. In F. Calimeri, G. Ianni, and M. Truszczynski, editors, *Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 9345 of *Lecture Notes in Computer Science*, pages 531–544. Springer, 2015.

[66] M. Gebser, M. Maratea, and F. Ricca. The design of the seventh answer set programming competition. In M. Balduccini and T. Janhunen, editors, *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 10377 of *Lecture Notes in Computer Science*, pages 3–9. Springer, 2017.

[67] M. Gebser, P. Obermeier, T. Schaub, M. Ratsch-Heitmann, and M. Runge. Routing driverless transport vehicles in car assembly with answer set programming. *Theory and Practice of Logic Programming*, 18(3-4):520–534, 2018.

[68] M. Gebser, T. Schaub, and S. Thiele. GrinGo: A new grounder for answer set programming. In C. Baral, G. Brewka, and J. S. Schlipf, editors, *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.

[69] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[70] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[71] E. Goldberg and Y. Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proceedings of the Design, Automation and Test in Europe Conference and Exposition*, pages 10886–10891. IEEE Computer Society, 2003.

[72] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2):140–174, 2003.

[73] D. Gunopulos, H. Mannila, and S. Saluja. Discovering all most specific sentences by randomized algorithms. In F. N. Afrati and P. G. Kolaitis, editors, *Proceedings of the 6th International Conference on Database Theory*, volume 1186 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1997.

[74] Gurobi Optimization, LLC. Gurobi optimizer. `www.gurobi.com`. Accessed: 2019-09-05.

[75] Y. Hamadi and C. M. Wintersteiger. Seven challenges in parallel SAT solving. *AI Magazine*, 34(2):99–106, 2013.

[76] G. H. Harman. The inference to the best explanation. *The Philosophical Review*, 74(1):88–95, 1965.

[77] F. Heras, A. Morgado, and J. Marques-Silva. MaxSAT-based encodings for group MaxSAT. *AI Communications*, 28(2):195–214, 2015.

[78] R. Hickey and F. Bacchus. Speeding up assumption-based SAT. In M. Janota and I. Lynce, editors, *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing*, volume 11628 of *Lecture Notes in Computer Science*, pages 164–182. Springer, 2019.

[79] J. R. Hobbs, M. E. Stickel, D. E. Appelt, and P. A. Martin. Interpretation as abduction. *Artificial Intelligence*, 63(1-2):69–142, 1993.

[80] J. N. Hooker. Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[81] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

[82] A. Hyttinen, F. Eberhardt, and M. Järvisalo. Constraint-based causal discovery: Conflict resolution with answer set programming. In N. L. Zhang and J. Tian, editors, *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 340–349. AUAI Press, 2014.

[83] A. Hyttinen, P. Saikko, and M. Järvisalo. A core-guided approach to learning optimal causal graphs: Paper supplement. `https://www.cs.helsinki.fi/group/coreo/dseptor/ijcai17-appendix.pdf`, 2017. Accessed: 2019-09-06.

[84] IBM. ILOG CPLEX Optimization Studio. `www.cplex.com`. Accessed: 2019-09-06.

[85] A. Ignatiev, A. Morgado, and J. Marques-Silva. Propositional abduction with implicit hitting sets. In G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, editors, *Proceedings of the 22nd European Conference on Artificial Intelligence*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1327–1335. IOS Press, 2016.

[86] A. Ignatiev, A. Morgado, G. Weissenbacher, and J. Marques-Silva. Model-based diagnosis with multiple observations. In S. Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 1108–1115. ijcai.org, 2019.

[87] A. Ignatiev, N. Narodytska, and J. Marques-Silva. Abduction-based explanations for machine learning models. *CoRR*, abs/1811.10656, 2018.

[88] A. Ignatiev, A. Previti, M. H. Liffiton, and J. Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In G. Pesant, editor, *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming*, volume 9255 of *Lecture Notes in Computer Science*, pages 173–182. Springer, 2015.

[89] Institut für Informatik, Universität Potsdam. Asparagus. `https://asparagus.cs.uni-potsdam.de/`. Accessed: 2019-09-05.

[90] M. Järvisalo, D. L. Berre, O. Roussel, and L. Simon. The international SAT solver competitions. *AI Magazine*, 33(1), 2012.

[91] M. Järvisalo, A. Biere, and M. Heule. Blocked clause elimination. In J. Esparza and R. Majumdar, editors, *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2010.

[92] M. Järvisalo, M. Heule, and A. Biere. Inprocessing rules. In B. Gramlich, D. Miller, and U. Sattler, editors, *Proceedings of the 6th International Joint Conference on Automated Reasoning*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012.

[93] M. Jose and R. Majumdar. Cause clue clauses: Error localization using maximum satisfiability. In M. W. Hall and D. A. Padua, editors, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 437–446. ACM, 2011.

[94] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[95] L. G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[96] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

[97] T. Korhonen, J. Berg, P. Saikko, and M. Järvisalo. MaxPre: An extended MaxSAT preprocessor. In S. Gaspers and T. Walsh, editors, *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017.

[98] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012.

[99] C. Lefèvre and P. Nicolas. The first version of a new ASP solver: ASPeRiX. In E. Erdem, F. Lin, and T. Schaub, editors, *Proceedings of the 10th International Conference on Logic Programming and*

Nonmonotonic Reasoning, volume 5753 of *Lecture Notes in Computer Science*, pages 522–527. Springer, 2009.

[100] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[101] C. M. Li and F. Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, chapter 19, pages 613–631. IOS Press, 2009.

[102] M. H. Liffiton, A. Previti, A. Malik, and J. Marques-Silva. Fast, flexible MUS enumeration. *Constraints*, 21(2):223–250, 2016.

[103] L. Magnani. Abductive reasoning: Philosophical and educational perspectives in medicine. In D. A. Evans and V. L. Patel, editors, *Advanced Models of Cognition for Medical Training and Practice*, pages 21–41. Springer, 1992.

[104] V. W. Marek and M. Truszczynski. Stable models and an alternative logic programming paradigm. *CoRR*, cs.LO/9809032, 1998.

[105] J. Marques-Silva, A. Ignatiev, and A. Morgado. Horn maximum satisfiability: Reductions, algorithms and applications. In E. C. Oliveira, J. Gama, Z. A. Vale, and H. L. Cardoso, editors, *Proceedings of the 18th EPIA Conference on Artificial Intelligence*, volume 10423 of *Lecture Notes in Computer Science*, pages 681–694. Springer, 2017.

[106] J. Marques-Silva, M. Janota, A. Ignatiev, and A. Morgado. Efficient model based diagnosis with maximum satisfiability. In Q. Yang and M. J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 1966–1972. AAAI Press, 2015.

[107] J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In D. Sciuto, editor, *Proceedings of Design, Automation and Test in Europe*, pages 408–413. IEEE Computer Society, 2008.

[108] J. P. Marques Silva and I. Lynce. On improving MUS extraction algorithms. In K. A. Sakallah and L. Simon, editors, *Proceedings of 14th International Conference on Theory and Applications of Satisfiability Testing*, volume 6695 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2011.

[109] J. P. Marques-Silva, I. Lynce, and S. Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability*, chapter 4, pages 131–153. IOS Press, 2009.

[110] J. P. Marques Silva and K. A. Sakallah. Conflict analysis in search algorithms for satisfiability. In *Proceedings of the Eighth International Conference on Tools with Artificial Intelligence*, pages 467–469. IEEE Computer Society, 1996.

[111] R. Martins, V. M. Manquinho, and I. Lynce. Open-WBO: A modular MaxSAT solver. In C. Sinz and U. Egly, editors, *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014.

[112] C. Mencía, A. Previti, and J. Marques-Silva. SAT-based Horn least upper bounds. In M. Heule and S. A. Weaver, editors, *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing*, volume 9340 of *Lecture Notes in Computer Science*, pages 423–433. Springer, 2015.

[113] E. Moreno-Centeno and R. M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Operations Research*, 61(2):453–468, 2013.

[114] A. Morgado, F. Heras, M. H. Liffiton, J. Planes, and J. Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[115] A. Morgado, A. Ignatiev, and J. Marques-Silva. MSCG: robust core-guided MaxSAT solving. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:129–134, 2014.

[116] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535. ACM, 2001.

[117] N. Narodytska and F. Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In C. E. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2717–2723. AAAI Press, 2014.

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

[119] P. Norvig. Inference in text understanding. In K. D. Forbus and H. E. Shrobe, editors, *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 561–565. Morgan Kaufmann, 1987.

[120] A. D. Palù, A. Dovier, E. Pontelli, and G. Rossi. Answer set programming with constraints using lazy grounding. In P. M. Hill and D. S. Warren, editors, *Proceedings of the 25th International Conference on Logic Programming*, volume 5649 of *Lecture Notes in Computer Science*, pages 115–129. Springer, 2009.

[121] M. Parker and J. Ryan. Finding the minimum weight IIS cover of an infeasible system of linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 17(1-2):107–126, 1996.

[122] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[123] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2 edition, 2009.

[124] C. S. Peirce. *Philosophical Writings of Peirce*. Dover, 1955.

[125] Y. Peng and J. A. Reggia. *Abductive inference models for diagnostic problem-solving*. Springer Science+Business Media, 1990.

[126] R. S. Peter Spirtes, Clark Glymour. *Causation, Prediction, and Search*. Adaptive Computation and Machine Learning. The MIT Press, 2 edition, 2001.

[127] S. Prestwich. CNF Encodings. In *Handbook of Satisfiability*, chapter 2, pages 75–97. IOS Press, 2009.

[128] A. Previti, A. Ignatiev, A. Morgado, and J. Marques-Silva. Prime compilation of non-clausal formulae. In Q. Yang and M. J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 1980–1988. AAAI Press, 2015.

[129] A. Previti and J. Marques-Silva. Partial MUS enumeration. In M. desJardins and M. L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press, 2013.

[130] K. Rantanen, A. Hyttinen, and M. Järvisalo. Learning optimal causal graphs with exact search. In M. Studený and V. Kratochvíl, editors, *Proceedings of the International Conference on Probabilistic Graphical Models*, volume 72 of *Proceedings of Machine Learning Research*, pages 344–355. PMLR, 2018.

[131] J. A. Reggia, D. S. Nau, and P. Y. Wang. Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, 19(5):437–460, 1983.

[132] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[133] R. Rymon. An SE-tree-based prime implicant generation algorithm. *Annals of Mathematics and Artificial Intelligence*, 11(1-4):351–366, 1994.

[134] K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger, and G. P. Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523–529, 2005.

[135] P. Saikko. Re-implementing and extending a hybrid SAT-IP approach to maximum satisfiability. Master's thesis, University of Helsinki, 2015.

[136] P. Saikko, J. Berg, and M. Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In N. Creignou and D. L. Berre, editors, *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016.

[137] P. Saikko, B. M. Malone, and M. Järvisalo. MaxSAT-based cutting planes for learning graphical models. In L. Michel, editor, *Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming*, volume 9075 of *Lecture Notes in Computer Science*, pages 347–356. Springer, 2015.

[138] P. Saikko, J. P. Wallner, and M. Järvisalo. AbHS: A propositional abduction solver. `https://www.cs.helsinki.fi/group/coreo/abhs/`, 2016. Accessed: 2019-09-09.

[139] K. Satoh and T. Uno. Enumerating minimal explanations by minimal hitting set computation. In J. Lang, F. Lin, and J. Wang, editors,

Proceedings of the First International Conference on Knowledge Science, Engineering and Management, volume 4092 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2006.

[140] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *INFORMS Journal on Computing*, 6(4):445–454, 1994.

[141] B. Schäpers, T. Niemueller, G. Lakemeyer, M. Gebser, and T. Schaub. ASP-based time-bounded planning for logistics robots. In M. de Weerdt, S. Koenig, G. Röger, and M. T. J. Spaan, editors, *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling*, pages 509–517. AAAI Press, 2018.

[142] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[143] D. A. Schum. Species of abductive reasoning in fact investigation in law. In M. MacCrimmon and P. Tillers, editors, *The Dynamics of Judicial Proof: Computation, Logic, and Common Sense*, volume 94 of *Studies in Fuzziness and Soft Computing*, pages 307–336. Physica, 2002.

[144] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[145] P. Spirtes. Directed cyclic graphical representations of feedback models. In P. Besnard and S. Hanks, editors, *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 491–498. Morgan Kaufmann, 1995.

[146] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques*. MIT press, 1986.

[147] R. T. Stern, M. Kalech, A. Feldman, and G. M. Provan. Exploring the duality in conflict-directed model-based diagnosis. In J. Hoffmann and B. Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press, 2012.

[148] J. Stillman. The complexity of propositional default logics. In W. R. Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 794–799. AAAI Press / The MIT Press, 1992.

[149] M. Studený. Bayesian networks from the point of view of chain graphs. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 496–503. Morgan Kaufmann, 1998.

[150] T. Syrjänen. Lparse 1.0 user's manual. `http://www.tcs.hut.fi/Software/smodels`, 2000. Accessed: 2019-09-06.

[151] S. Triantafillou and I. Tsamardinos. Constraint-based causal discovery from multiple interventions over overlapping variable sets. *Journal of Machine Learning Research*, 16:2147–2205, 2015.

[152] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. H. Siekmann and G. Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, Symbolic Computation, pages 466–483. Springer Berlin Heidelberg, 1983.

[153] T. Uno and K. Satoh. Detailed description of an algorithm for enumeration of maximal frequent sets with irredundant dualization. In B. Goethals and M. J. Zaki, editors, *Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[154] J. N. M. van Loon. Irreducibly inconsistent systems of linear inequalities. *European Journal of Operational Research*, 8(3):283–288, 1981.

[155] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[156] K. Yamamoto, T. Onishi, and Y. Tsuruoka. Hierarchical reinforcement learning with abductive planning. *CoRR*, abs/1806.10792, 2018.

[157] L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the Design, Automation and Test in Europe Conference and Exposition*, pages 10880–10885. IEEE Computer Society, 2003.

[158] C. S. Zhu, G. Weissenbacher, and S. Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In P. Bjesse and A. Slobodová, editors, *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, pages 63–66. FMCAD Inc., 2011.