Master's thesis

Master's Programme in Computer Science

Pseudo-Boolean Optimization by Implicit Hitting Sets

Pavel Smirnov

February 14, 2022

Faculty of Science

University of Helsinki

Supervisor(s)

Prof. Matti Järvisalo, Dr. Jeremias Berg

Examiner(s)

Prof. Matti Järvisalo, Dr. Jeremias Berg

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Study programme Master's Programme in Computer Science

Tekijä — Författare — Author

Pavel Smirnov

Työn nimi — Arbetets titel — Title

Pseudo-Boolean Optimization by Implicit Hitting Sets

Ohjaajat — Handledare — Supervisors

Prof. Matti Järvisalo, Dr. Jeremias Berg

Työn laji — Arbetets art — Level Master's thesis	Aika — Datum — Month and year February 14, 2022	Sivumäärä — Sidoantal — Number of pages 63 pages, 2 appendix pages

Tiivistelmä — Referat — Abstract

There are many computationally difficult problems where the task is to find a solution with the lowest cost possible that fulfills a given set of constraints. Such problems are often NP-hard and are encountered in a variety of real-world problem domains, including planning and scheduling. NP-hard problems are often solved using a declarative approach by encoding the problem into a declarative constraint language and solving the encoding using a generic algorithm for that language. In this thesis we focus on pseudo-Boolean optimization (PBO), a special class of integer programs (IP) that only contain variables that admit the values 0 and 1.

We propose a novel approach to PBO that is based on the implicit hitting set (IHS) paradigm, which uses two separate components. An IP solver is used to find an optimal solution under an incomplete set of constraints. A pseudo-Boolean satisfiability solver is used to either validate the feasibility of the solution or to extract more constraints to the integer program. The IHS-based PBO algorithm iteratively invokes the two algorithms until an optimal solution to a given PBO instance is found.

In this thesis we lay out the IHS-based PBO solving approach in detail. We implement the algorithm as the PBO-IHS solver by making use of recent advances in reasoning techniques for pseudo-Boolean constraints. Through extensive empirical evaluation we show that our PBO-IHS solver outperforms other available specialized PBO solvers and has complementary performance compared to classical integer programming techniques.

ACM Computing Classification System (CCS)
Mathematics of computing → Discrete mathematics → Combinatorics → Combinatorial optimization
Theory of computation → Logic → Constraint and logic programming

Avainsanat — Nyckelord — Keywords

constraint optimization, pseudo-Boolean optimization, implicit hitting sets

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms study track

Preface

This work has been conducted in the Constraint Reasoning and Optimization research group starting from late 2020 until the end of 2021 while being financially supported by Academy of Finland under grants 322869 and 328718. I would like to thank Matti Järvisalo and Jeremias Berg for providing invaluable guidance and advice both with respect to writing this thesis as well as developing the PBO-IHS solver. I would also like to acknowledge and thank Paul Saikko for his initial implementation of the PBO-IHS solver. His work in implementing a baseline version of PBO-IHS provided a starting point for research and implementation work that eventually led to this thesis. Computational resources were provided by Finnish Grid and Cloud Infrastructure (urn:nbn:fi:research-infras-2016072533).

Contents

1 Introduction

There are many computationally difficult problems where the task is to find a solution that fulfills a given set of constraints. Going further, for many constrained problems the goal is to find an optimal solution, meaning that there is no better solution to the problem according to a given objective. Optimization problems are often solved to reduce costs and to improve the efficiency and productivity of a variety of industrial and societal operations. Such problems are encountered in industrial development, e.g., when optimizing the cost of quality assurance of aircrafts [16], or configuring a set of virtual machines to reduce energy consumption while increasing the workload [81]. The cost of transportation is reduced by solving optimization problems in the fields of logistics [51, 54] and city planning [17]. Allocating human resources by scheduling work shifts [2, 7], sports matches [36] and course lectures [12] are also optimization problems that are solved to maximize productivity while preventing undesirable outcomes, such as double-booking events, causing long work shifts or scheduling short travel and rest times. Optimization problems are also encountered when identifying genetic variations [27, 52], performing cryptanalysis [78, 53], implementing circuits [69, 101, 100] and allocating radar stations [98], among various other domains.

In this thesis, we focus on the declarative approach to solving optimization problems. In the declarative approach, an instance of a constrained optimization problem is represented in a declarative language via an encoding. Instead of developing an algorithm to solve problem instances from a specific domain, in the declarative approach an algorithm is developed that solves problem instances that are encoded in a specific declarative language. Various optimization paradigms have been developed, including mixed integer programming (MIP) [77], answer set programming (ASP) [48], Boolean satisfiability (SAT) [44] based maximum satisfiability (MaxSAT) [11] and its extensions to e.g. optimization modulo theories and MaxSMT [21, 91]. It is notable that many such languages can be used to succinctly encode instances of NP-hard problems [11, 91]. Despite the computational complexity, many solvers that have been developed to tackle problems encoded in such languages [25, 3, 14, 37, 9] are used to solve real-world problems efficiently.

We focus on pseudo-Boolean optimization (PBO) as the declarative language in this thesis. In PBO the constraints of a given problem are encoded as a set of pseudo-Boolean (PB) constraints [84], which are linear inequalities over binary variables with integer coefficients. A solution to a PBO instance is an assignment of the binary variables that fulfills the set of linear inequalities while minimizing a linear cost function over the same variables.

Many PBO solving approaches have been proposed. As PBO is a special case of MIP and instances of PBO are also known as 0–1 integer programs, instances of PBO can be solved using a MIP solver [25, 3, 14]. Another approach is to translate a PBO instance into a MaxSAT instance and to solve the resulting MaxSAT instance with a MaxSAT solver [73, 90]. Multiple algorithms use reasoning techniques directly for pseudo-Boolean constraints [67, 41, 20, 32, 93] to find a solution that satisfies the given set of PB constraints. To find an optimal solution, a common technique is to run the algorithm iteratively, where the upper bound on the cost of a solution that decreases with each iteration is encoded as a PB constraint [67, 90]. An optimization technique that uses the so-called core-guided search has also been proposed [33].

The main contribution of this thesis is a novel approach for solving instances of PBO by making use of implicit hitting sets (IHS) [85, 89]. In an IHS-based approach, a minimum-cost hitting set (MCHS) solver is used to optimize the solution, and a separate oracle is used to validate its feasibility. For our IHS-based PBO solver, the MCHS solver is implemented with a MIP solver that uses a branch-and-cut algorithm (see e.g. [5]) and the oracle is implemented with a pseudo-Boolean solver that uses direct reasoning for pseudo-Boolean constraints [41, 32]. IHS has proven itself to be succesful with solving instances of optimization problems modeled in other declarative languages such as MaxSAT [29, 31, 30, 87], ASP [88] and MaxSMT [43]. IHS has also been succesfully applied in solving other types of constrained problems [85, 89, 56, 55, 57].

In this thesis, we present the IHS-based PBO solving algorithm and detail the implementation of the algorithm called PBO-IHS. To our knowledge, this approach to solving PBO is novel in that we are not aware of earlier work studying the applicability of IHS in the context of PBO. We harness recent advances in direct reasoning techniques for pseudo-Boolean constraints by adapting the pseudo-Boolean solver called Roundingsat [41, 32] as the oracle component in PBO-IHS. We present multiple practical search techniques to be used in conjunction with the IHS-based PBO solving algorithm to improve the solving capability of PBO-IHS. We provide results from an extensive empirical evaluation of PBO-IHS, comparing its performance with a range of earlier developed specialized solvers for PBO as well as a commercial MIP solver, and evaluate the impact of the various search techniques on the empirical performance of PBO-IHS. From the evaluation we find that

PBO-IHS outperforms other available specialized PBO solvers and has complementary performance compared to classical integer programming techniques. Parts of the results presented in this thesis have been published and presented at the 27th International Conference on Principles and Practice of Constraint Programming (CP 2021) [94].

The rest of the text is organized as follows. Preliminary concepts are defined in Section 2. In Section 3 there is an overview of other approaches to solving PBO. The theoretical groundwork for the IHS-based PBO solving algorithm is layed out in Section 4. In Section 5 multiple practical techniques that were considered for PBO-IHS to improve the implementation of the algorithm are detailed. Section 6 details experiments conducted with the available PBO solvers to evaluate and analyze the performance of PBO-IHS in comparison to other PBO solvers. Finally, the work is concluded and future work is outlined in Section 7.

2 Preliminaries

Preliminary concepts that are required for this work are overviewed in this section. In Section 2.1, the linear pseudo-Boolean optimization (PBO) problem is defined. Linear program (LP) relaxations of integer programs (IP) are discussed in Section 2.2. Lastly, concepts that are relevant to the implicit hitting set (IHS) methodology are defined and explained in Sections 2.3 and 2.4.

2.1 Pseudo-Boolean optimization

A binary variable x has the domain $\{0, 1\}$. A *literal* l over x is either x itself or its *negation*, $\overline{x} \equiv (1 - x)$. The negation of the negation of a literal is the literal itself, $\overline{(\overline{x})} = 1 - \overline{x} = 1 - (1 - x) = x$. An *assignment* is a function $\tau : X \to \{0, 1\}$, where X is a set of binary variables. When convenient we treat an assignment τ over X as a set of literals

$$\tau = \{x \mid x \in X \land \tau(x) = 1\} \cup \{\overline{x} \mid x \in X \land \tau(x) = 0\}.$$

Conversely, a set of literals τ for which $l \in \tau \Rightarrow \bar{l} \notin \tau$ is treated as an assignment for which $\tau(l) = 1 \Leftrightarrow l \in \tau$.

A *pseudo-Boolean (PB) constraint* C is a linear inequality of the form $\sum_i a_i l_i \geq b$, where $a_i, b \in \mathbb{Z}$ and l_i are literals. A PB constraint is in normalized form if the variables that appear in the PB constraint are distinct and the coefficients a_i and bound b are non-negative. Any linear inequality of binary variables can be rewritten in *normalized form* using standard algebraic manipulation.

The set of variables that are present in C is denoted by $\textsc{var}(C)$. An assignment $\tau : X \to \{0, 1\}$ satisfies C ($\tau(C) = 1$) if $\textsc{var}(C) \subset X$ and $\sum_i a_i \tau(l_i) \geq b$. A *PB formula* $F \equiv \{C_1, C_2, ..., C_n\}$ is a set of PB constraints. We denote $\textsc{var}(F) = \bigcup_i \textsc{var}(C_i)$. An assignment $\tau : \textsc{var}(F) \to \{0, 1\}$ is a solution to F ($\tau(F) = 1$) if $\tau(C_i) = 1$ for every $C_i \in F$.

Example 2.1. The linear inequality $x_1 - 1 \leq x_2 - 2 \cdot x_3$ can be rewritten in normalized form as follows:

$$\begin{array}{rcll}
x_1 - 1 & \leq & x_2 - 2 \cdot x_3 & \Leftrightarrow \\
x_2 - 2 \cdot x_3 & \geq & x_1 - 1 & \Leftrightarrow \\
-x_1 + x_2 - 2 \cdot x_3 & \geq & -1 & \Leftrightarrow \\
-(1 - \overline{x_1}) + x_2 - 2 \cdot (1 - \overline{x_3}) & \geq & -1 & \Leftrightarrow \\
-1 + \overline{x_1} + x_2 - 2 + 2 \cdot \overline{x_3} & \geq & -1 & \Leftrightarrow \\
\overline{x_1} + x_2 + 2 \cdot \overline{x_3} & \geq & 2 &
\end{array}$$

△

Example 2.2. Consider the PB formula

$$\begin{aligned}
F = \{x_1 + 2 \cdot x_2 + 3 \cdot \overline{x_3} & \geq 3, \\
x_1 + \overline{x_2} & \geq 1, \\
3 \cdot \overline{x_2} + 5 \cdot \overline{x_3} & \geq 4\}.
\end{aligned}$$

Now, τ_A for which $\tau_A(x_1) = 1$ $(\tau_A(\overline{x_1}) = 0)$, $\tau_A(x_2) = 1$, $\tau_A(x_3) = 0$ $(\tau_A(\overline{x_3}) = 1)$ is a solution to F because it satisfies all three constraints:

$$\begin{aligned}
1 + 2 \cdot 1 + 3 \cdot 1 = 6 & \geq 3, \\
1 + 0 = 1 & \geq 1, \\
3 \cdot 0 + 5 \cdot 1 = 5 & \geq 4.
\end{aligned}$$

On the other hand, τ_B for which $\tau_B(x_1) = 0, \tau_B(x_2) = 1, \tau_B(x_3) = 0$ does not satisfy the second constraint because $0 + 0 = 0 \not\geq 1$. Hence τ_B is not a solution to F. △

Example 2.3. Consider the PB formula

$$\begin{aligned}
F = \{x_1 + 2 \cdot x_2 + 3 \cdot \overline{x_3} & \geq 3, \\
\overline{x_1} + x_2 & \geq 2, \\
3 \cdot \overline{x_2} + 5 \cdot \overline{x_3} & \geq 8\}.
\end{aligned}$$

To satisfy the third constraint, an assignment τ must have $\tau(x_2) = \tau(x_3) = 0$. But if $\tau(x_2) = 0$, then the second constraint is not satisfied regardless of the value of $\tau(x_1)$:

$$\begin{aligned}
\tau(x_1) = 1 & \Rightarrow 0 + 0 = 0 \not\geq 2, \\
\tau(x_1) = 0 & \Rightarrow 1 + 0 = 1 \not\geq 2.
\end{aligned}$$

Hence no τ satisfies all three constraints of F, i.e., F has no solutions. △

An instance of the pseudo-Boolean *decision problem* is a PB formula F. The task is to determine if a solution to F exists, in which case F is *satisfiable* (`SAT`). If no solutions to F exist, then F is *unsatisfiable* (`UNSAT`). Sometimes we determine satisfiability of F under a set of *assumptions* A, where A is a set of literals. F is satisfiable under an assumption set A if there exists a solution τ such that $\tau(F) = 1$ and $\tau(l) = 1$ for each $l \in A$.

Example 2.4. Consider the PB formula F from Example 2.2. The formula F is satisfiable under the assumption set $\{x_1, \overline{x_3}\}$ because τ_A from Example 2.2 is a solution to F with $\tau_A(x_1) = \tau_A(\overline{x_3}) = 1$. The formula F is not satisfiable under the assumption set $\{x_3\}$. There is no assignment τ with $\tau(x_3) = 1$ that satisfies the third constraint because $3 \cdot \tau(\overline{x_2}) + 5 \cdot 0 \geq 4$ does not hold regardless of the value of $\tau(\overline{x_2})$. △

An instance of the pseudo-Boolean optimization (PBO) problem consists of a PB formula F and a linear *cost function* $\mathrm{O} \equiv \sum_i w_i l_i$, where each $w_i \in \mathbb{Z}^+$ and each l_i is a literal over some $x_i \in \textsc{var}(F)$. We will sometimes abuse notation and treat O as a set of tuples, where $(w_i, l_i) \in \mathrm{O}$ if and only if $w_i l_i$ is a term in O. The cost function O is expressed in normalized form. Therefore if $(w_i, l_i) \in \mathrm{O}$, then $(w', \overline{l_i}) \notin \mathrm{O}$ for any w'. The set of variables that are present in O is $\textsc{var}(\mathrm{O})$. We specify

$$\textsc{lit}(\mathrm{O}) = \{l \mid l \in \textsc{var}(\mathrm{O})\} \cup \{\bar{l} \mid l \in \textsc{var}(\mathrm{O})\}.$$

If an assignment $\tau : X \to \{0, 1\}$ is such that $\textsc{var}(\mathrm{O}) \subset X$, then the cost of τ is $\mathrm{O}(\tau) = \sum_i w_i \tau(l_i)$. The cost of a literal l is

$$\mathrm{O}(l) = \begin{cases} w & \text{if } (w, l) \in \mathrm{O}, \\ 0 & \text{otherwise.} \end{cases}$$

Because O is in normalized form, $\mathrm{O}(l) > 0$ if and only if $\mathrm{O}(\bar{l}) = 0$ for each $l \in \textsc{var}(\mathrm{O})$. In the pseudo-Boolean optimization problem, the task is to find an optimal solution, which is a solution τ to F such that $\mathrm{O}(\tau) \leq \mathrm{O}(\tau')$ for every solution τ' to F.

Example 2.5. Consider a PBO instance consisting of the PB formula F from Example 2.2 and a cost function $\mathrm{O} = x_1 + 2 \cdot \overline{x_2} + 3 \cdot \overline{x_3}$. Because there are three variables, there are $2^3 = 8$ possible assignments. As shown in Example 2.4, an assignment τ such that $\tau(x_3) = 1$ is not a solution to F. There are 4 assignments that set x_3 to 0, one of which is τ_B from Example 2.2 that is also not a solution to F. The remaining 3 assignments are solutions to F, which have the following costs:

- $\tau_1(x_1) = 0$, $\tau_1(x_2) = 0$, $\tau_1(x_3) = 0$, $\mathrm{O}(\tau_1) = 0 + 2 \cdot 1 + 3 \cdot 1 = 5$,
- $\tau_2(x_1) = 1$, $\tau_2(x_2) = 0$, $\tau_2(x_3) = 0$, $\mathrm{O}(\tau_2) = 1 + 2 \cdot 1 + 3 \cdot 1 = 6$,
- $\tau_3(x_1) = 1$, $\tau_3(x_2) = 1$, $\tau_3(x_3) = 0$, $\mathrm{O}(\tau_3) = 1 + 2 \cdot 0 + 3 \cdot 1 = 4$.

An optimal solution (the only optimal solution for this specific PBO instance) is τ_3. $\triangle$

The Boolean SAT problem [44] is a well-known NP-complete problem [22]. SAT is a special case of the PB decision problem because every clause expressed in conjunctive normal form (as a CNF clause) can be expressed as a PB constraint. Finding an optimal solution is at least as hard as finding any solution. Therefore PBO is NP-hard.

Example 2.6. Consider the CNF clause $x_1 \vee x_2 \vee \neg x_3$. An assignment τ does not violate the clause if $\tau(x_1) = 1$, $\tau(x_2) = 1$, or $\tau(x_3) = 0$. This is expressed as the PB constraint $(x_1 + x_2 + (1 - x_3) \geq 1)$. $\triangle$

2.2 Integer programs and linear relaxations

Integer programming (see e.g. [79]) is a declarative approach for solving NP-hard optimization problems. In this work we consider integer programs that consist of a linear cost function and a set of linear inequalities as constraints. Unlike instances of PBO that only admit the values 0 and 1 as variable assignments, a solution to an *integer program* (IP) can contain general integers, i.e., $\tau(x_i) \in \mathbb{Z}$ for a solution τ to the given IP.

The *linear program (LP) relaxation* of a given IP has the same cost function and the same set of constraints, but allows for the variables to be assigned to a real number in the solution, i.e., $\tau(x_i) \in \mathbb{R}^+$ for a solution τ to the LP relaxation. The cost of an optimal solution to an IP is at least the cost of an optimal solution to its LP relaxation, because every solution to the IP is a solution to its LP relaxation. For a solution τ to the LP relaxation of a 0–1 IP (i.e. PBO instance), the assignment to variables is bounded between zero and one, i.e., $\tau(x_i) \in [0, 1]$.

Example 2.7. Consider the IP with the cost function $\mathrm{O} = x_1 + x_2$ and the constraints $F = \{(2x_1 + x_2 \geq 1/2), (-x_2 \geq -3/2), (-x_1 + 2x_2 \geq 1/2)\}$. There are two variables in the IP, allowing the solution space of the IP to be illustrated in a two-dimensional graph in Figure 2.1. All solutions to the IP are bounded inside the triangle that is formed from the three lines representing the three constraints of the IP. The space bounded by the constraints contains exactly two integer solutions at (0,1) and (1,1). The point at (0,1) represents the optimal solution to the IP because the cost of (0,1) is less than the cost of (1,1). The LP relaxation of the IP, on the other hand, admits non-integer solutions. This means that any point contained within the triangle represents a solution to the LP relaxation. The blue rectangle at $(0.1, 0.3)$ represents a solution to the LP relaxation that is not a solution to the IP. The cost of $(0.1, 0.3)$ is lower than the cost of the optimal solution to the IP $(0.1 + 0.3 < 1)$, meaning that the optimal solutions of the LP relaxation have a smaller cost than the optimal solution to the IP.

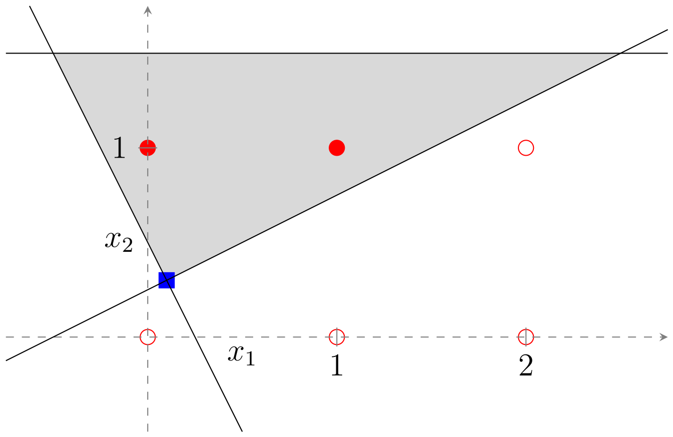


Figure 2.1: Visual representation of the integer program from Example 2.7.

△

2.3 Unsatisfiable cores

The concept of unsatisfiable cores is central to this thesis because the IHS-based PBO solver presented in this work relies on efficiently identifying cores of a given PBO instance. For an instance of PBO with a PB formula F and a cost function O, a set of literals $\kappa \subset \textsc{lit}(\mathrm{O})$ is an *(unsatisfiable) core* if it fulfills the following conditions:

i) any assignment τ for which $\tau(l) = 0$ for every $l \in \kappa$ is not a solution to F,

ii) if $l \in \kappa$, then $\bar{l} \notin \kappa$.

Therefore a solution τ to F must assign at least one literal in each core of the PBO instance to 1.

Example 2.8. Consider the PBO instance from Example 2.5. An assignment τ for which $\tau(x_1) = \tau(\overline{x_2}) = 0$ is not a solution to F because it does not satisfy the constraint $(x_1 + \overline{x_2} \geq 1)$. Therefore $\{x_1, \overline{x_2}\}$ is a core of F. △

2.4 Minimum-cost hitting set

In this thesis we consider the minimum-cost hitting set problem (adapted from [61]) in the context of unsatisfiable cores of a given PBO instance.

Definition 2.9. Let $K = \{\kappa_1, \kappa_2, \ldots, \kappa_k\}$ be a set of unsatisfiable cores of a PBO instance F with a cost function O. A set $H \subset \textsc{lit}(\mathrm{O})$ is a *hitting set* over K if

i) $H \cap \kappa \neq \emptyset$ for each $\kappa \in K$, and

ii) if $l \in H$, then $\bar{l} \notin H$.

In Figure 2.2 (adapted from [85]), an example of a hitting set is visualized. Suppose the PBO instance for which the hitting set is found has the cost function $\mathrm{O} = \sum_{i=1}^{10} l_i$. The elements in $\textsc{lit}(\mathrm{O})$ are visualized as circles and the cores in K are visualized as rectangles. The negations of the literals are omitted because the set K does not have cores that contain $\overline{l_i}$ for any $i \in \{1, \ldots, 10\}$. The circles that are colored gray form a hitting set H.

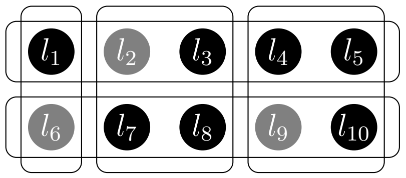


Figure 2.2: Visual representation of a hitting set (adapted from [85]).

In the minimum-cost hitting set (MCHS) problem, given a cost function O and a set of cores K, the goal is to find a minimum-cost hitting set over K. The cost of a hitting set is $\mathrm{O}(H) = \sum_{l \in H} \mathrm{O}(l)$. A *minimum-cost* hitting set is a hitting set $H \subset \textsc{lit}(\mathrm{O})$ over K such that $\mathrm{O}(H) \leq \mathrm{O}(H')$ for all hitting sets $H' \subset \textsc{lit}(\mathrm{O})$ over K.

Definition 2.10. Suppose $H \subset \textsc{lit}(\mathrm{O})$ such that H meets the condition (ii) of Definition 2.9. Let $H^C := \{l \in \textsc{lit}(\mathrm{O}) \mid l, \bar{l} \notin H\}$. An *extension* of H is

$$\gamma_H := H \cup \{l \mid l \in H^C \wedge \mathrm{O}(l) = 0\}.$$

The extension γ_H of H meets the condition (ii) because if $l \in H^C$, then $l, \bar{l} \notin H$ and $\mathrm{O}(l) = 0$ if and only if $\mathrm{O}(\bar{l}) > 0$. If H is a hitting set over K, then so is γ_H because if $H \subset \gamma$, then $\gamma \cap \kappa \neq \emptyset$ for each $\kappa \in K$. The extension of a hitting set is refered to as the *hitting set assignment*. Including literals in $\{l \mid l \in H^C \wedge \mathrm{O}(l) = 0\}$ does not incur additional cost. This means that the hitting set assignment γ_H is an assignment of all variables in $\textsc{var}(\mathrm{O})$ such that γ_H has the same cost as H. Therefore if H is a minimum-cost hitting set, then γ_H is also a minimum-cost hitting set. We will use the terms hitting set and hitting set assignment interchangeably when it is clear from context.

An instance of the MCHS problem can be encoded as a 0–1 IP, where the cost function of the IP is O and for each core $\kappa \in K$ the IP has the constraint $(\sum_{l \in \kappa} l \geq 1)$. The solution to the IP is a minimum-cost hitting set assignment over K.

Example 2.11. Consider the instance of the MCHS problem that is visualized in Figure 2.2. The IP that encodes the MCHS instance has the cost function $\mathrm{O} = \sum_{i=1}^{10} x_i$ and the constraint set $F = \{(l_1 + l_6 \geq 1), (l_2 + l_3 + l_7 + l_8 \geq 1), (l_4 + l_5 + l_9 + l_{10} \geq 1), (l_1 + l_2 + l_3 + l_4 + l_5 \geq 1), (l_6 + l_7 + l_8 + l_9 + l_{10} \geq 1)\}$. An optimal solution τ is the hitting set assignment represented by the gray circles, such that $\tau(l_i) = 1$ for $i \in \{2, 6, 9\}$ and $\tau(l_i) = 0$ otherwise. $\triangle$

3 An overview of PBO algorithms

Before discussing the PBO-IHS solver, we survey other approaches to solving PBO, some of which are compared empirically in their performance against PBO-IHS in Section 6. The PBO-IHS solver makes use of some of the algorithms presented in this section. We start with Section 3.1 by discussing the branch-and-cut algorithm for solving IP. In Section 3.2 we discuss how instances of PBO can be encoded to CNF and solved through the use of a SAT solver. Finally, we describe PBO solving algorithms that use reasoning techniques for pseudo-Boolean constraints directly.

3.1 IP solving with branch-and-cut

Since PBO is a special case of IP, PBO instances can be solved with an IP solver. Many state-of-the-art IP solvers such as IBM CPLEX [25] and SCIP [3, 14] implement the so-called branch-and-cut algorithm (see e.g. [5]). The branch-and-cut algorithm first solves the LP relaxation of a given IP instance. If an optimal solution to the LP relaxation contains variable assignments that are non-integer, the search space is branched into multiple disjoint search spaces which are then solved separately as subproblems. The subproblems are LPs themselves that are made disjoint using a branching constraint. Branching is done recursively on the resulting subproblems, until the subproblem contains an integer optimal solution or it contains no integer solutions at all. An integer optimal solution of a subproblem becomes a solution candidate for the IP, in which case the branch-and-cut algorithm proceeds to find a lower cost integer solution to the subproblems that were not yet processed. Cutting refers to the method of restricting the subproblem using cutting planes derived from non-integer solutions to the subproblem. The purpose of cutting is to prune out some of the solutions that are either non-integer or non-optimal from the subproblem.

Solving LP relaxations is fast because there exist algorithms for solving LPs that have polynomial worst-case time complexity [62, 60, 35, 15]. The Simplex algorithm (see e.g. [92]) is one of the most commonly used algorithms for solving LPs in state-of-the-art IP solvers [25, 3, 14]. Despite the fact that Simplex has an exponential worst-case time complexity with respect to the number of variables and constraints [63], smoothed analysis shows that Simplex solves most LPs in polynomial time [95].

3.2 PBO solving by encoding to CNF

There are several approaches to encoding a set of PB constraints into a set of CNF clauses. The approach presented in [73] uses the so-called generalized totalizer encoding [58], while another approach presented in [90] uses binary decision diagrams [1]. There are also different approaches to finding an optimal solution. In [73] the cost function is encoded as a set of unit soft clauses as presented in [70], encoding the PBO instance into a MaxSAT instance that is then solved using a MaxSAT solver. The algorithm presented in [90] obtains upper and lower bounds on the cost of an optimal solution. The upper bound is expressed as a CNF clause, while the lower bound is implemented through the use of assumptions. The resulting SAT formula is solved iteratively with a SAT solver, tightening the bounds with each iteration.

3.3 Repurposing CDCL for PB solving

The conflict-driven clause learning (CDCL) algorithm [72, 103] is an algorithm implemented in most state-of-the-art SAT solvers [37, 9]. CDCL can also be extended to reason with PB constraints directly. The extended CDCL algorithm makes *decisions* on how certain variables are assigned to specific values, which often lead to *propagating* assignments for other variables due to the structure of the constraints. If decisions and propagations lead to a *conflict* in the PB instance, the conflict is analyzed in order to derive (i.e., learn) a conflict constraint, which is added into the set of constraints. The worst-case exponential running time of the extended CDCL algorithm comes from the fact that there can be an exponential number of decisions that the algorithm can make at any point during the execution. In order to satisfy a learnt conflict constraint, the algorithm is restricted from making a set of decisions that cannot lead to a solution to the PB instance, accelerating the solving process.

The PB algorithm presented in [67] has two distinct ways in which it learns a conflict constraint. The first way is through resolution, which is a reasoning technique that is widely used in SAT solvers. In [67], the resolution rule is used to learn conflict constraints by treating PB constraints as clauses. The second way is by implementing cutting planes [23] based on techniques described in [20] that rely on the so-called weakening and saturation rules. A more recent implementation of a PB algorithm [41] uses the so-called division and rounding rules instead of weakening and saturation rules. In theory, for a given PB instance, the proof of unsatisfiability using cutting planes is never longer than the proof of unsatisfiability using resolution. However, there exist instances for which the proof of unsatisfiability using resolution is exponentially longer. In practice, it is challenging to implement a CDCL-based PB algorithm that uses cutting planes in order to learn conflict constraints efficiently.

The PB solving algorithm presented in [32] uses the LP relaxation of the PB formula to learn conflict constraints. At certain points during the solving process, the LP relaxation is solved with the variables fixed according to the decisions and propagations set by the extended CDCL algorithm. An LP solver is used to determine whether the LP relaxation is unsatisfiable (i.e., rationally infeasible) given the decisions and propagations, which means that the PB instance itself is unsatisfiable under such decisions and propagations. A conflict constraint is derived by analyzing the Farkas multipliers [42] extracted from the rationally infeasible LP instance. Since solving LP can be done in polynomial time [95], deriving a conflict constraint by determining rational infeasibility of the LP can often be faster than deriving the same conflict constraint through decisions in the extended CDCL algorithm.

The CDCL-based PB algorithms discussed so far can be extended to optimization via a *solution-improving search* (refered to as linear search in [33], sequential search in [90] and strengthening in [67]). Suppose that an algorithm has found a solution τ_i to the PBO instance with a cost function O. In solution-improving search, the constraint $(\sum_{(w_i,l_i)\in \mathrm{O}} w_i l_i < \mathrm{O}(\tau_i))$ is added to the set of constraints. If the PBO instance is still satisfiable, then the algorithm finds a solution τ_{i+1} for which $\mathrm{O}(\tau_{i+1}) < \mathrm{O}(\tau_i)$. The process is repeated until the algorithm returns `UNSAT` when there is no solution with lower cost than the solution found at the previous iteration.

Another way of extending a PB algorithm to optimization as presented in [33] is by extending the so-called OLL algorithm [8] to PBO solving. In short, the PB formula is iteratively solved under different assumption sets. If the formula is satisfiable, a constraint is added that bounds the cost of the solution from above, similar to solution-improving search. If the formula is unsatisfiable under an assumption set, the learnt conflict constraint is processed. Literals that are not in the assumption set are removed from the conflict constraint using the so-called weakening rule. The non-unit coefficients in the constraint are rounded to have a unit coefficient. The cost function is then reformulated based on the constraint, so that it allows one more of the literals in the cost function to be set to 1 in subsequent iterations. The optimization procedure alternates between this method and using the solution-improving search.

4 Solving PBO via implicit hitting sets

In this section we present how implicit hitting sets (IHS) are used to solve instances of PBO. The correctness of the IHS-based PBO algorithm is proven by adapting proofs originally presented for IHS-based MaxSAT solving in [29]. In Section 4.1, we show that computing a minimum-cost hitting set over all cores of the given PBO instance leads to solving the PBO instance itself. Because enumerating every unsatisfiable core of a given PBO instance is impractical, in Section 4.2 we describe how a minimum-cost hitting set is computed over an incomplete set of cores using implicit hitting sets. Lastly, in Section 4.3 we present a rudimentary version of the IHS-based PBO algorithm in pseudocode.

4.1 Solving PBO by computing a minimum-cost hitting set over all cores

Let K be the set of all cores of a given PBO instance with a PB formula F and a cost function O. For a solution τ to F the function restriction is $\tau\restriction_{\textsc{lit}(\mathrm{O})}\colon \textsc{var}(\mathrm{O}) \to \{0,1\}$ for which $\tau\restriction_{\textsc{lit}(\mathrm{O})}(l) = \tau(l)$ for each $l \in \textsc{lit}(\mathrm{O})$. A literal that is not in $\textsc{lit}(\mathrm{O})$ cannot incur cost. Therefore $\mathrm{O}(\tau\restriction_{\textsc{lit}(\mathrm{O})}) = \mathrm{O}(\tau)$. Now consider a set $H \subset \textsc{lit}(\mathrm{O})$ for which $l \in H \Rightarrow \bar{l} \notin H$. The extension γ_H (recall Definition 2.10) is an assignment of literals in $\textsc{lit}(\mathrm{O})$, which means γ_H is a function restriction ($\gamma_H = \tau'\restriction_{\textsc{lit}(\mathrm{O})}$) for some assignment $\tau' : \textsc{var}(F) \to \{0,1\}$.

Proposition 4.1. If an assignment $\tau : \textsc{var}(F) \to \{0,1\}$ is a solution to F, then $\tau\restriction_{\textsc{lit}(\mathrm{O})}$ is a hitting set over K.

Proof. By definition of unsatisfiable cores, if τ is a solution to F, then for any unsatisfiable core κ, $\tau(l) = 1$ for at least one $l \in \kappa$. When viewed as a set, $\tau \cap \kappa \neq \emptyset$ for all $\kappa \in K$ and $l \in \tau \Rightarrow \bar{l} \notin \tau$. Because $\kappa \subset \textsc{lit}(\mathrm{O})$ for every $\kappa \in K$, $\tau\restriction_{\textsc{lit}(\mathrm{O})}$ is a hitting set over K. □

We now show that a minimum-cost hitting set H can be used to find an optimal solution to F.

Proposition 4.2. Suppose F is a PB formula and O is a cost function. For a minimum-cost hitting set H over all unsatisfiable cores K, there exists an optimal solution τ to F such that $\tau\restriction_{\text{LIT(O)}}= \gamma_H$, where γ_H is the hitting set assignment of H.

Proof. First we prove that there exists a solution τ to F such that $\tau\restriction_{\text{LIT(O)}}= \gamma_H$. We do this through contradiction by supposing the opposite, which means that for every solution τ to F there is at least one literal l for which $\tau(l) \neq \gamma_H(l)$. We show that the set $\kappa = \{l \mid \gamma_H(l) = 0\}$ is an unsatisfiable core in such case. If $\gamma_H(l) = 0$, then $\gamma_H(\bar{l}) \neq 0$, which means $l \in \kappa \Rightarrow \bar{l} \notin \kappa$ holds. If $\tau(l) = 0$ for all $l \in \kappa$, then τ is not a solution to F unless $\tau\restriction_{\text{LIT(O)}}= \gamma_H$. Therefore κ is an unsatisfiable core and $\kappa \in K$. Because γ_H is a hitting set over K, then $\gamma_H(l) = 1$ for at least one $l \in \kappa$, which is a contradiction.

Next we prove that τ is an optimal solution, i.e., there is no solution τ^* to F for which $\text{O}(\tau^*) < \text{O}(\tau)$. Suppose opposite is true. Because τ^* is a solution to F, then $\tau^*\restriction_{\text{LIT(O)}}$ is a hitting set over K by Proposition 4.1. We now have $\text{O}(\tau^*\restriction_{\text{LIT(O)}}) = \text{O}(\tau^*) < \text{O}(\tau) = \text{O}(\tau\restriction_{\text{LIT(O)}}) = \text{O}(\gamma_H) = \text{O}(H)$, which is a contradiction.

□

It is important to note that while we have proven the existence of an optimal solution where $\tau(l) = \gamma_H(l)$ for all $l \in \text{LIT(O)}$, an assignment for variables in $\text{VAR}(F) \setminus \text{VAR(O)}$ has to be found through other means.

4.2 Implicit hitting sets

Computing a minimum-cost hitting set over K leads to solving the given PBO instance, if K contains all unsatisfiable cores of that PBO instance. Explicitly enumerating every core is impractical, because the number of unsatisfiable cores for the given PBO instance can, in the worst case, be exponential with respect to the number of literals in the cost function. If κ is a core, then any $\kappa' \subset \text{LIT(O)}$ such that $\kappa' \supset \kappa$ is also a core. Going further, there exist instances of PBO, where the number of cores that are not subsets of other cores is exponential. We demonstrate this with an example that is adapted from [30].

Example 4.3. Suppose $n \in \mathbb{N}$ is even, $F_n = \{\sum_{i=1}^{n} l_i \geq n/2\}$ and $\text{O} = \sum_{i=1}^{n} l_i$. There is no solution to F_n that assigns more than $n/2$ literals l_i to 0, but any assignment that assigns $n/2$ literals l_i to 1 is a solution. Therefore if $\kappa \subset \{l_1, \dots, l_n\}$ and $|\kappa| = n/2 + 1$, then κ is a core and any set $\kappa' \subsetneq \kappa$ is not a core. The number of such cores is $\binom{n}{n/2+1}$, which is exponential in n. $\triangle$

In the implicit hitting set (IHS) approach a minimum-cost hitting set over all cores K is found by iteratively computing a hitting set over an incomplete set $K_i \subset K$, where K_i at each iteration i contains increasingly more cores, i.e., $K_i \subsetneq K_{i+1}$. There exists an iteration k for which $K_k = K$. Hence a minimum-cost hitting set H over K will eventually be found. The idea in the IHS approach is to find H at an iteration $i < k$.

The IHS approach requires an *oracle* that fulfills two requirements. First, the oracle must be able to verify if a given set H is a hitting set over the complete set of cores K. The second requirement is that in case the oracle determines that H is not a hitting set over K, then the oracle must provide at least one core $\kappa \in K$ that H did not hit. There exist PB solvers that fulfill the requirements of an oracle by solving the PB formula F under a set of assumptions A.

The PB solver fulfills the first requirement of an oracle because it can verify if the hitting set assignment γ_H of H is a hitting set over K by determining if F under an assumption set γ_H returns `SAT` and a solution τ. In that case $\tau \restriction_{\text{LIT(O)}} = \gamma_H$ is a hitting set over K by Proposition 4.1.

If solving F under an assumptions set γ_H returns `UNSAT`, then there is no solution τ to F such that $\tau \restriction_{\text{LIT(O)}} = \gamma_H$, which means γ_H is not a hitting set over K. The PB solver also extracts a subset $\kappa \subset \{\bar{l} \mid l \in \gamma_H\}$ such that F under an assumption set $\{\bar{l} \mid l \in \kappa\}$ is unsatisfiable. Because there is no solution to F for which $\tau(l) = 0$ for each $l \in \kappa$, κ is an unsatisfiable core for which $\gamma_H \cap \kappa = \emptyset$. Going forward we will refer to a PB solver that fulfills the requirements of an oracle as the *PB oracle*.

The outline of the IHS approach is as follows. Starting from $i := 0$ and $K_0 := \emptyset$, a minimum-cost hitting set H_i over K_i is computed and the PB oracle is queried to determine if H_i is a hitting set over K. If the PB oracle verfies that H_i is not a hitting set and provides a core κ, then a minimum-cost hitting set H_{i+1} over $K_{i+1} := K_i \cup \{\kappa\}$ is computed and the PB oracle is queried again with H_{i+1}. The process is repeated until the PB oracle verifies that H_i is a hitting set over K in some iteration i.

Because $K_i \subsetneq K_{i+1}$, every (minimum-cost) hitting set over K_{i+1} is a hitting set over K_i, but not every (minimum-cost) hitting set over K_i is a hitting set over K_{i+1}. Therefore $\text{O}(H_i) \leq \text{O}(H_{i+1})$ and for some $j \leq k$, $\text{O}(H_j) = \text{O}(\tau)$ where τ is an optimal solution to F. The cost of H_i is a lower bound on the cost of an optimal solution τ because $\text{O}(H_i) \leq \text{O}(\tau)$

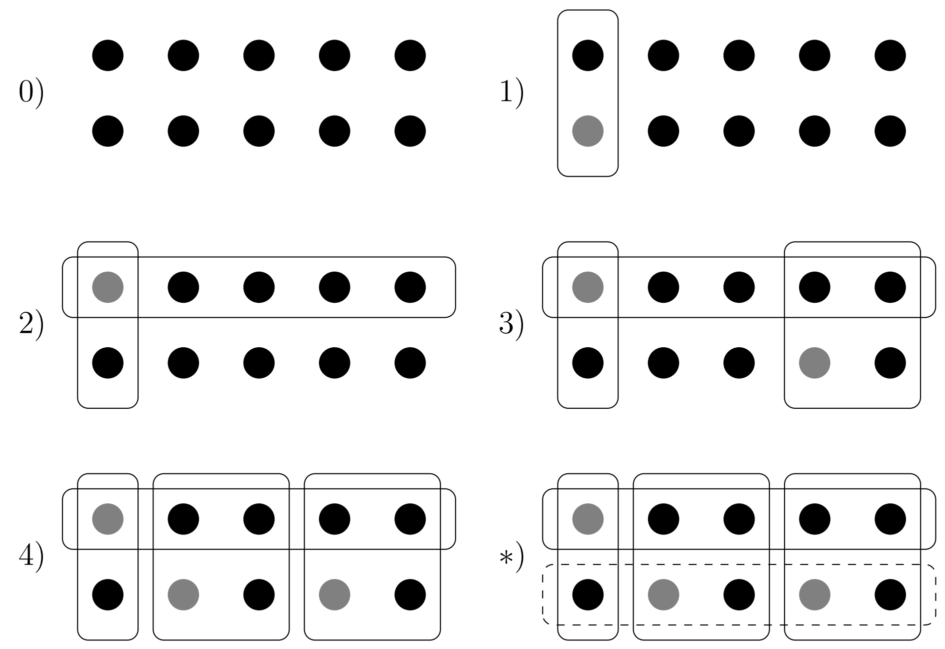


Figure 4.1: Illustration of the IHS approach (adapted from [85]).

for any i.

The IHS approach is illustrated in Figure 4.1, where the MCHS problem visualized in Figure 2.2 is solved in four iterations. For each Subfigure (i), the rectangles represent the cores in K_i. Gray circles represent literals that are in the minimum-cost hitting set computed for K_i. The Subfigure ($*$) illustrates the full set of cores K, with Subfigures (0-4) illustrating how a new core is introduced with each subsequent K_i. Notice that the minimum-cost hitting set computed in the fourth iteration is a minimum-cost hitting set over the full set K. A set that is missing from K_4 is highlighted in Subfigure ($*$) with a dashed line rectangle.

4.3 A rudimentary algorithm for PBO using IHS

A rudimentary algorithm for solving PBO with IHS uses two solvers as black-box interfaces: an MCHS solver and a PB oracle. We use MCHS(O, K) to represent a call to the MCHS solver to compute a minimum-cost hitting set assignment γ over K with respect to the cost function O, where K is a set of IP constraints that encodes the MCHS instance. We use PBSolve(F, A) to represent a call to the PB oracle to solve F under an assumption set A. The PBSolve(F, A) call returns a tuple (sat?, κ, τ), where sat? specifies whether the result was `SAT` or `UNSAT`. In case sat? is `UNSAT`, τ is empty and $\kappa \subset \{\bar{l} \mid l \in A\}$

Algorithm 1: The rudimentary IHS-based PBO solving algorithm.

```
Input  : A PB formula $F$ and a cost function O
Output: SAT/UNSAT, an optimal solution $\tau$
1  (sat?, $\kappa$, $\tau$) := PBSolve($F$, $\emptyset$);
2  if $sat?$ = UNSAT then                        // Check that $F$ has solutions
3      return (UNSAT, $\emptyset$);
4  [$UB$ := O($\tau$), $\tau_{best}$ := $\tau$];
5  $K$ := $\emptyset$;
6  while $TRUE$ do
7      $\gamma$ := MCHS(O, $K$);
8      [$LB$ := O($\gamma$)];
9      (sat?, $\kappa$, $\tau$) := PBSolve($F$, $\gamma$);
10     if $sat?$ = SAT then
11         return (SAT, $\tau$);
12     else
13         $K := K \cup \{(\sum_{l \in \kappa} l \geq 1)\}$;
```

is an unsatisfiable core that the PB oracle has extracted. In case sat? is SAT, κ is empty and τ is a solution to F that was found as proof of satisfiability.

The pseudocode of the rudimentary IHS-based PBO algorithm is presented in Algorithm 1. Algorithm 1 performs the first call to the PB oracle with an empty assumption set at Line 1 to make sure that F has solutions from which to find an optimal solution. Next, the algorithm proceeds to the main loop, where it alternates between calling both black-box solvers. Starting with an empty set of constraints $K = \emptyset$ (Line 5), the MCHS solver returns a hitting set assignment γ over cores encoded in K at Line 7.

At Line 10, if the PB oracle finds a solution τ to F under the assumption set γ, Algorithm 1 is terminated and τ is returned as an optimal solution to F. If no such solution exists, the unsatisfiable core κ is encoded as a constraint and added to K at Line 13. The MCHS solver is then called again with updated K, iterating the main loop.

The optional Line 8 computes the lower bound LB of the cost of an optimal solution. The lower bound can be output periodically during the execution of the IHS-based PBO algorithm or returned if the user halts the algorithm prematurely. The lower bound is computed from the cost of a minimum-cost hitting set of K at each iteration. As

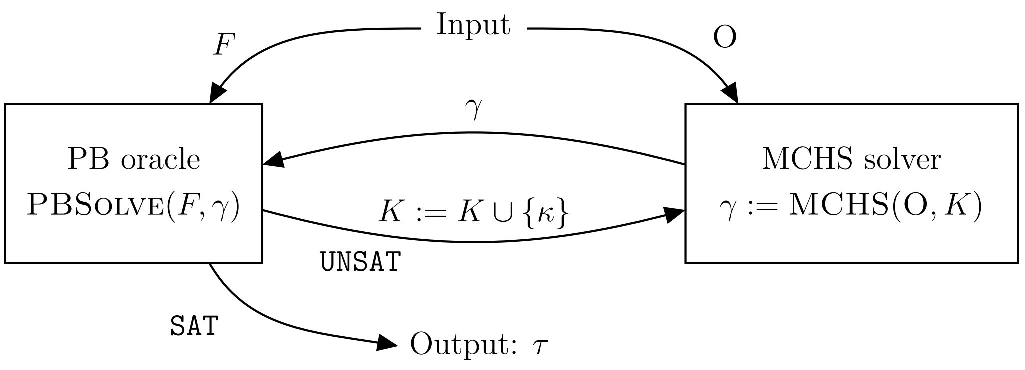


Figure 4.2: The IHS-based PBO solving algorithm (adapted from [85]).

new constraints are added to the existing set of constraints K, LB cannot decrease in subsequent iterations. The optional Line 4 computes the upper bound UB from the solution τ that was obtained after verifying that the PBO instance is satisfiable, then stores τ as τ_{best} as the best solution encountered so far. Bounding an optimal solution and storing the best encountered solution becomes more relevant when discussing some of the techniques that improve upon the rudimentary IHS-based PBO algorithm in Section 5.

Figure 4.2 illustrates the flow of execution of Algorithm 1. Given an input (F, O), the PB oracle is given F as input and the MCHS solver is given O as input. The main loop starting from Line 6 is illustrated by the two arrows between the two boxes that illustrate the PB oracle and the MCHS solver. The result of the MCHS solver γ (Line 7) is supplied to the PB oracle as an assumption set (Line 9). If the result of the PB oracle is `UNSAT`, κ is supplied to the MCHS solver by encoding κ as a constraint that is added to K (Line 13). If the result of the PB oracle is `SAT`, τ is returned as output (Line 11).

Theorem 4.4. *Given a PBO instance with a PB formula F and a cost function O, Algorithm 1 terminates and returns an optimal solution τ to F.*

Proof. First we show that τ returned by Algorithm 1 is an optimal solution. The assignment τ is a solution to F because it is returned by the PB oracle. The hitting set assignment γ returned by the MCHS solver is an assignment for all literals in $\textsc{lit}(\mathrm{O})$ such that $\mathrm{O}(\gamma)$ is at most the cost of an optimal solution to F. Now $\tau \restriction_{\textsc{lit}(\mathrm{O})} = \gamma$, because τ is a solution to F under an assumption set γ. Therefore $\mathrm{O}(\tau) = \mathrm{O}(\tau \restriction_{\textsc{lit}(\mathrm{O})}) = \mathrm{O}(\gamma)$, which means τ is an optimal solution.

Next we show that Algorithm 1 terminates. There is a finite number of cores for the given PBO instance. Line 13 ensures that the number of cores encoded in K increases with each iteration. We prove that the PB oracle does not return the same core twice.

Suppose a core $\kappa_i = \{l_1, ..., l_q\}$ is extracted at iteration i, which means that a constraint $(l_1+...+l_q \geq 1)$ is added to K. If $\text{PBS}\textsc{olve}(F, \gamma)$ extracts the same core in iteration $j > i$, then $\gamma \supset \{\bar{l} \mid l \in \kappa_i\}$. This is a contradiction because γ returned in iteration j violates the constraint $(l_1 + ... + l_q \geq 1)$. Therefore, in the worst case, the halting condition at Line 11 is met after k iterations, where k is the number of all cores in the PBO instance. □

In terms of computational complexity, there are three potential sources of exponential runtimes. Firstly, an instance of MCHS is solved at every iteration, where MCHS is in general an NP-hard problem [61]. Secondly, the PB oracle extracts a core by solving an NP-complete PB decision problem. Finally, there are PBO instances such as Example 4.3 for which the IHS-based PBO algorithm has to find an exponential number of cores with respect to $|\textsc{var}(\text{O})|$ before terminating. This was shown in [29] for the IHS-based MaxSAT algorithm and the proof can be adapted for the IHS-based PBO algorithm as well. As with other algorithms that are capable of solving instances of NP-hard problems, often the worst case runtimes can be avoided when solving real-world problem instances. Therefore empirical evaluation of the performance of such algorithms is often more informative than the analysis of their computational complexity.

Example 4.5. Consider a PBO instance with $F = \{\sum_{i=1}^{5} l_i \geq 2\}$ and $\text{O} = \sum_{i=1}^{5} l_i$. An assignment that assigns exactly two arbitrary literals to 1 is an optimal solution.

The PBO instance passes the check at Line 2, because τ for which $\tau(l_i) = 1$ for each $i = 1, 2, .., 5$ is a solution ($UB = 5$). The first $\text{MCHS}(\text{O}, K)$ call on an empty constraint set K finds a minimum-cost hitting set of cost zero ($LB = 0$), where $\gamma(l_i) = 0$ for each $i = 1, 2, .., 5$. The $\text{PBS}\textsc{olve}(F, \gamma)$ call returns `UNSAT` because the assignment τ for which $\tau(l_i) = 0$ for each $i = 1, 2, .., 5$ is not a solution to F. There are multiple choices for a core that the PB oracle could return, suppose $\kappa = \{l_1, l_2, l_3, l_4\}$. After the first $\text{PBS}\textsc{olve}(F, \gamma)$ call, the constraint set is $K = \{(l_1 + l_2 + l_3 + l_4 \geq 1)\}$.

The second $\text{MCHS}(\text{O}, K)$ call has multiple choices for γ that it can return, suppose $\gamma(l_1) = 1$ and $\gamma(l_i) = 0$ for $i = 2, .., 5$ ($LB = 1$). The $\text{PBS}\textsc{olve}(F, \gamma)$ call returns `UNSAT` because the assignment τ for which $\tau(l_1) = 1$ and $\tau(l_i) = 0$ for $i = 2, .., 5$ is not a solution to F. The core that is returned is $\kappa = \{l_2, l_3, l_4, l_5\}$ and the constraint set is updated to $K = \{(l_1 + l_2 + l_3 + l_4 \geq 1), (l_2 + l_3 + l_4 + l_5 \geq 1)\}$.

Suppose the third $\text{MCHS}(\text{O}, K)$ call returns $\gamma(l_2) = 1$ and $\gamma(l_i) = 0$ for $i \neq 2$ ($LB = 1$). The $\text{PBS}\textsc{olve}(F, \gamma)$ call still returns `UNSAT`, the returned core is $\kappa = \{l_1, l_3, l_4, l_5\}$ and the constraint set is updated to $K = \{(l_1+l_2+l_3+l_4 \geq 1), (l_2+l_3+l_4+l_5 \geq 1), (l_1+l_3+l_4+l_5 \geq$

$1)\}$. A similar pattern of core extraction continues. For each $i = 1, ..., 5$, a core κ is extracted such that $\kappa = \{l_j \mid 1 \leq j \leq 5\} \setminus \{l_i\}$. Finally when $K = \{(l_1 + l_2 + l_3 + l_4 \geq 1), (l_2 + l_3 + l_4 + l_5 \geq 1), (l_1 + l_3 + l_4 + l_5 \geq 1), (l_1 + l_2 + l_4 + l_5 \geq 1), (l_1 + l_2 + l_3 + l_5 \geq 1)\}$, the next $\mathrm{MCHS}(\mathrm{O}, K)$ call returns a hitting set with two literals assigned to 1. Suppose $\gamma(l_1) = \gamma(l_2) = 1$ and $\gamma(l_i) = 0$ for $i = 3, .., 5$ ($LB = 2$). An assignment τ for which $\tau(l_i) = \gamma(l_i)$ for $i = 1, .., 5$ is a solution to F. Therefore PBSolve(F, γ) returns `SAT`, halting the algorithm and returning τ as an optimal solution. $\triangle$

5 Improvements to IHS-based PBO solving

In this section we discuss several techniques considered for the PBO-IHS solver to improve upon the rudimentary IHS-based PBO solving algorithm presented in Algorithm 1. Techniques that reduce the number of MCHS solver and PB oracle calls or reduce the time taken by them can significantly impact the overall solving time of PBO-IHS.

In Section 5.1 we briefly discuss subset-minimization, a technique that decreases the size of an extracted core via additional calls to the PB oracle. In Section 5.2 we discuss how shuffling the assumption set can be used to extract multiple cores from a single call to the PB oracle. We restructure the IHS-based PBO algorithm to include a so-called disjoint phase to extract multiple cores in between calls to the MCHS solver in Sections 5.3 and 5.4. In Section 5.5 we describe a way to offload some of the information of the PBO instance onto the MCHS solver to produce hitting sets that violate fewer PB constraints.

In Section 5.6 we show how solving for hitting sets that are not minimum-cost is enough to implement PBO-IHS, as long as the cost of the hitting sets is below the cost of the best solution found so far. In Section 5.7 we discuss techniques that allow PBO-IHS to fix variables to specific values during the execution of PBO-IHS.

5.1 Core shrinking through additional PB calls

As shown in [29] in the context of MaxSAT, extracting smaller cores improves the performance of the IHS algorithm. Suppose the PB oracle returns a core κ. Using additional queries to the PB oracle it is possible to find a smaller core $\kappa' \subsetneq \kappa$. To perform core-shrinking, we make use of a subset-minimization algorithm (see e.g. [71]). Given a core κ, the core-shrinking algorithm excludes one literal l^* from κ and calls on the PB oracle with an assumption set $A = \{\bar{l} \mid l \in \kappa \setminus \{l^*\}\}$. If the PB oracle returns `SAT`, then the excluded literal is put back into κ and the process is repeated with another literal excluded. However, if the PB oracle returns `UNSAT` and a core κ', then $l^* \notin \kappa' \subset \kappa \setminus \{l^*\}$, in which case κ is updated to $\kappa := \kappa'$. The algorithm halts when each literal in κ has been excluded from the assumption set exactly once. Although not implemented for the purposes of this thesis, a carefully chosen stopping heuristic to the subset-minimization algorithm could be used to balance the number of additional PB calls with the size of the shrinked core.

Example 5.1. Let $F = \{(\sum_{i=1}^{5} l_i \geq 3), (l_1 + l_3 \geq 1)\}$. Suppose the PB oracle extracts a core $\kappa = \{l_1, l_2, l_3\}$ during the execution of the PBO-IHS algorithm. The set κ is a core because an assignment τ for which $\tau(l_1) = \tau(l_2) = \tau(l_3) = 0$ violates the constraint $(\sum_{i=1}^{5} l_i \geq 3)$. The core-shrinking algorithm calls $\text{PBS}\textsc{olve}(F, \{\bar{l} \mid l \in \kappa \setminus \{l_1\}\})$. The PB oracle returns `SAT` because there is a solution τ to F such that $\tau(l_2) = \tau(l_3) = 0$ and $\tau(l_i) = 1$ for $i \notin \{2, 3\}$. The core-shrinking algorithm keeps the l_1 in κ and proceeds to call the PB oracle with l_2 excluded from κ. The $\text{PBS}\textsc{olve}(F, \{\bar{l} \mid l \in \kappa \setminus \{l_2\}\})$ call returns `UNSAT` because an assignment τ for which $\tau(l_1) = \tau(l_3) = 0$ violates the constraint $(l_1 + l_3 \geq 1)$. The PB oracle returns a core $\kappa' := \{l_1, l_3\}$, the core-shrinking algorithm updates $\kappa := \kappa'$ and calls the PB oracle the third time with $\text{PBS}\textsc{olve}(F, \{\bar{l} \mid l \in \kappa \setminus \{l_3\}\})$. PB oracle returns `SAT` because an assignment τ for which $\tau(l_i) = 1$ for $i = 2, 3, 4, 5$ is a solution to F. After three additional PB oracle calls, the subset-minimization algorithm obtains a smaller core $\kappa = \{l_1, l_3\}$. △

5.2 Assumption set shuffling

When the $\text{PBS}\textsc{olve}(F, A)$ is called, the CDCL-based PB solver assigns the value to each literal in A one at a time. While the satisfiability of the PB formula remains the same regardless of the order of the literals in A, the PB solver may arrive at a different contradiction from a different subset of literals in A as shown in [86] in the context of MaxSAT. Therefore it is possible that the core κ returned from an `UNSAT` call to the PB oracle is different depending on the order of literals in A.

Once the PB decision solver finds that F is unsatisfiable while assigning literals from the assumption set A, the sequence of literals in A is permuted at random and the literals of A are assigned again in the order of the permuted sequence. The process is repeated for some predefined number of iterations, with each iteration potentially resulting in a different core.

Example 5.2. Let $F = \{(l_1 + l_2 + l_3 \geq 2), (l_3 + l_4 + l_5 \geq 1)\}$ and $A = \{\overline{l_1}, \overline{l_2}, \overline{l_3}, \overline{l_4}, \overline{l_5}\}$. The PB formula F is unsatisfiable under the assumption set A. If the order of A starts with $(\overline{l_1}, \overline{l_3})$, then the first constraint is violated after assigning two literals, resulting in core $\{l_1, l_3\}$. If the order of A starts with $(\overline{l_3}, \overline{l_4}, \overline{l_5})$, then the second constraint is violated after assigning three literals, resulting in core $\{l_3, l_4, l_5\}$. △

Given a set of distinct cores extracted using assumption shuffling, various heuristics can be used to choose appropriate cores from the set. One heuristic is to include all distinct cores that are not supersets of other cores in the set into the MCHS instance. This allows for multiple cores to be extracted in between calls to the MCHS solver.

Another heuristic that we consider is to only include the smallest core. When applied to the PB formula F and the assumption set A from Example 5.2, if the two cores from the example are extracted, the heuristic includes $\{l_1, l_3\}$ to the MCHS solver but does not include $\{l_3, l_4, l_5\}$. This heuristic provides an alternative method to extracting smaller cores compared to subset-minimization discussed in Section 5.1. Core shrinking relies on multiple calls to the PB oracle that can be exponential in their running time in the worst case. On the other hand, reassigning literals in A during the assumption set shuffling is always done in polynomial time. In contrast to core shrinking, there is no guarantee that the resulting core will be the smallest possible when using assumption set shuffling.

5.3 Disjoint cores

Instead of extracting one core at a time in between calls to the MCHS solver, multiple disjoint cores can be extracted by manipulating a single hitting set. In the context of MaxSAT solving [29] it was proposed to extract multiple disjoint cores in the initial stage of the IHS algorithm, after which the algorithm would proceed to extract one core in between calls to the MCHS solver. However, in [86] it was found that extracting multiple disjoint cores at every iteration of the IHS algorithm improves performance of solving MaxSAT. We adapt this idea to PBO solving. Algorithm 2 shows in pseudocode how disjoint cores are extracted.

In comparison to Algorithm 1, Algorithm 2 introduces a new loop at Line 12 inside the main loop, which we will refer to as the *disjoint phase*. Starting from $A := \gamma$, after extracting a core κ, the algorithm removes literals whose negations are present in κ from the assumption set A at Line 14. This causes the next core extracted at Line 15 to be disjoint with every previous core extracted during the disjoint phase. Before every PBSOLVE(F, A) call during the disjoint phase, the assumption set A is made strictly smaller at Line 14. This means that eventually the PB oracle returns `SAT`, exiting the loop at Line 12. As verified at Line 2, the PB oracle returns `SAT` under an empty assumption set. However, the PB oracle may return `SAT` under a non-empty assumption set as well.

Algorithm 2: IHS-based PBO solving algorithm with disjoint cores.

Input : A PB formula F and a cost function O

Output: `SAT`/`UNSAT`, an optimal solution τ

```
1  (sat?, $\kappa$, $\tau$) := PBSolve($F, \emptyset$);
2  if $sat? =$ UNSAT then                          // Check that $F$ has solutions
3      return (UNSAT, $\emptyset$);
4  $UB :=$ O($\tau$), $\tau_{best} := \tau$, $K := \emptyset$;
5  while $TRUE$ do
6      $\gamma :=$ MCHS(O, $K$);
7      $LB :=$ O($\gamma$);
8      if $LB = UB$ then
9          return (SAT, $\tau_{best}$);
10     $A := \gamma$;
11     (sat?, $\kappa$, $\tau$) := PBSolve($F, A$);
12     while $sat? =$ UNSAT do                     // Disjoint phase
13         $K := K \cup \{\sum_{l \in \kappa} l \geq 1\}$;
14         $A := A \setminus \{\bar{l} \mid l \in \kappa\}$;
15         (sat?, $\kappa$, $\tau$) := PBSolve($F, A$);
16     if $O(\tau) < UB$ then
17         $UB :=$ O($\tau$), $\tau_{best} := \tau$;
18     if $LB = UB$ then
19         return (SAT, $\tau_{best}$);
```

As opposed to the rudimentary IHS-based PBO algorithm, during the execution of Algorithm 2 the PB oracle may return `SAT` multiple times. Once the assumption set A is reduced in Line 14 after extracting at least one core, the solution τ under A is no longer such that $\tau \restriction_{\text{LIT}(\mathrm{O})} = \gamma$. Therefore τ is not guaranteed to be an optimal solution, which means that the algorithm cannot be halted when τ is returned.

Finding solutions to F during the execution of PBO-IHS allows for obtaining an upper bound on the cost of an optimal solution (UB). At Line 16, the cost of τ obtained by a `SAT` call to the PB oracle is compared to the cost of the previously stored solution τ_{best}. In case the cost of τ is lower than the cost of τ_{best}, τ is set as τ_{best}, maintaining a record of the lowest cost solution that was found by PBO-IHS so far.

The lower bound on the cost of an optimal solution (LB) is obtained from the cost of a minimum-cost hitting set obtained at Line 7. Instead of terminating when the PB solver returns `SAT`, Algorithm 2 terminates when the lower and the upper bounds meet. At Line 8, the bounds are compared after calling the MCHS solver, because it is possible that the lower bound is made higher by the minimum-cost hitting set returned at Line 6. Should the lower and upper bounds meet at Line 8, there is no need to call the PB oracle to retrieve a solution, because $\mathrm{O}(\tau_{best})$ is exactly the lower bound on the cost of an optimal solution, in which case τ_{best} is an optimal solution. Notice that γ obtained at Line 6 may not be the restriction of τ_{best} to $\text{LIT}(\mathrm{O})$.

At Line 18 the bounds are checked after the disjoint phase, in case the upper bound was tightened due to finding a lower cost solution τ_{best}. This halting condition also covers the halting condition of the rudimentary IHS-based PBO algorithm. Suppose τ is a solution to F under the assumption set $A = \gamma$ that is obtained at Line 10. In that case Algorithm 2 skips the entire disjoint phase. Because $\gamma = \tau \restriction_{\text{LIT}(\mathrm{O})}$, then $\mathrm{O}(\tau) = \mathrm{O}(\gamma) = LB$. The cost of τ is stored as UB at Line 17, after which the condition at Line 18 is fulfilled, terminating the algorithm.

Maintaining both the upper and lower bound provides valuable information that can be used to evaluate the cost of an optimal solution, for example, in case PBO-IHS needs to be prematurely interrupted by the user. PBO-IHS with disjoint cores also maintains τ_{best} as a witness to the upper bound, which can act as an intermediate result.

In addition to guiding the PB oracle to extract disjoint cores, another benefit to introducing the disjoint phase is that the IHS-based PBO algorithm makes less calls to the MCHS solver to extract the same number of cores. We conclude this section by simulating Algorithm 2 for a few iterations on a concrete PBO instance in Example 5.3.

Example 5.3. Consider a PBO instance with $F = \{\sum_{i=1}^{5} l_i \geq 4\}$ and $\mathrm{O} = \sum_{i=1}^{5} l_i$. Clearly an assignment that assigns exactly four literals to 1 is an optimal solution.

Suppose the first call to the PB oracle at Line 2 returns a solution τ for which $\tau(l_i) = 1$ for each $i = 1, 2, .., 5$. The upper bound UB is set to 5 and τ is designated as τ_{best}. The first $\mathrm{MCHS}(\mathrm{O}, K)$ call on an empty constraint set K returns γ for which $\gamma(l_i) = 0$ for each $i = 1, 2, .., 5$ and $LB := 0$. Bounds do not meet, $LB \neq UB$ because $0 \neq 5$. The algorithm proceeds to the disjoint phase. Assumption set A initially contains all literals that are in γ. The $\textsc{PBSolve}(F, A)$ call returns `UNSAT` and, as there are multiple choices for cores to return, suppose the PB oracle returns $\kappa = \{l_1, l_2\}$. The set κ is a core because an assignment τ such that $\tau(l_1) = \tau(l_2) = 0$ can only assign at most three literals to 1, violating the constraint in F. The core κ is encoded as a constraint and added to K, at which point $K = \{(l_1 + l_2 \geq 1)\}$.

Next, the assumption set A is modified so that all negated literals from κ are subtracted: $A := A \setminus \{\overline{l_1}, \overline{l_2}\} = \{\overline{l_3}, \overline{l_4}, \overline{l_5}\}$. The $\textsc{PBSolve}(F, A)$ call returns `UNSAT` again, and the disjoint phase proceeds to its second iteration. Suppose the obtained core is $\kappa = \{l_3, l_4\}$. Adding κ to K produces $K = \{(l_1 + l_2 \geq 1), (l_3 + l_4 \geq 1)\}$.

The negated literals from κ are again subtracted from A, leaving $A = \{\overline{l_5}\}$. Calling the PB oracle under the assumption set $\{\overline{l_5}\}$ returns `SAT` with solution τ for which $\tau(l_5) = 0$ and $\tau(l_i) = 1$ for each $i = 1, 2, 3, 4$. This concludes the disjoint phase. The cost of τ is 4, which is lower than $UB = 5$. Hence τ is designated as τ_{best} and the upper bound is refined to $UB := 4$. We know already that τ_{best} is an optimal solution. However, the algorithm still has more work to do to prove this, since $LB = 0 \neq 4 = UB$. Therefore the algorithm proceeds to the second iteration of the main loop.

Let the next call to the MCHS solver return γ for which $\gamma(l_i) = 1$ if $i \in \{1, 3\}$ and $\gamma(l_i) = 0$ otherwise. The halting condition at Line 8 fails because $LB = 2 \neq 4 = UB$. Therefore the algorithm proceeds to the disjoint phase. Let $\textsc{PBSolve}(F, A = \gamma)$ return $\kappa = \{l_2, l_4\}$. Notice that the extracted cores are only disjoint with other cores that were extracted within the same disjoint phase. Now, $K = \{(l_1 + l_2 \geq 1), (l_3 + l_4 \geq 1), (l_2 + l_4 \geq 1)\}$ and $A = \{l_1, l_3, \overline{l_5}\}$. The $\textsc{PBSolve}(F, A)$ call returns `SAT` with the same solution as the one at the end of the previous disjoint phase.

The main loop of Algorithm 2 is repeated until enough cores are extracted for K so that the cost of a minimum-cost hitting set is 4. Let the MCHS solver return γ such that $\gamma(l_1) = 0$ and $\gamma(l_i) = 1$ for $i = 2, .., 5$. The algorithm has already stored a solution τ_{best} for which $\tau_{best}(l_5) = 0$ and $\tau_{best}(l_i) = 1$ for $i = 1, .., 4$, which also has the cost 4. The halting condition at Line 8 is fulfilled, which spares the algorithm from calling the PB oracle with $A = \gamma$ as an assumption set, returning τ_{best} as an optimal solution. Notice that $\gamma \neq \tau_{best} \restriction_{\text{LIT}(\text{O})}$.

5.4 Weight-aware core extraction

Weight-aware core extraction (WCE) is a technique first proposed in the context of core-guided MaxSAT solving in [13] and later explored in the context of PBO under the name independent cores in [33]. In the context of IHS, WCE offers a generalization to the disjoint phase. Instead of forcing all cores extracted within the same disjoint phase to be disjoint, literals with larger coefficients in the cost function may be shared among several cores. The goal is to increase the number of cores that is extracted within the same disjoint phase.

Algorithm 3 details the IHS-based PBO algorithm with the WCE technique in pseudocode. Lines that were added or modified with respect to Algorithm 2 are outlined with a comment included on the right side. At Lines 10 and 11 before the start of a disjoint phase, a variable $W(l) := \text{O}(l)$ is initialised for every literal $l \in \text{LIT}(\text{O})$. We will refer to $W(l)$ as the working weight of literal l. After extracting a core κ during an iteration of the disjoint phase, the minimum working weight w_{min} of the literals in κ is computed at Line 15. Then, each literal from κ has its working weight subtracted by w_{min}. At Line 18, instead of subtracting the negation of every literal of κ from the assumption set, the algorithm only subtracts the negation of a literal if its working weight is set to zero. Notice that at least one literal will be subtracted from the assumption set on each iteration, making it so the modified disjoint phase terminates eventually. However, because it is not necessarily the case that $W(l) = 0$ for all $l \in \kappa$, not all literals in κ have their negations subtracted from the assumption set. Therefore more cores may be extracted within the disjoint phase with WCE compared to the disjoint phase without WCE.

Example 5.4. Suppose a PBO instance with $F = \{(l_1 + l_N \geq 1), (l_2 + l_N \geq 1), ..., (l_n + l_N \geq 1)\}$ and $\text{O} = \sum_{i=1}^{n} l_i + nl_N$. Every core of this PBO instance contains the literal l_N, since any assignment τ for which $\tau(l_N) = 1$ is a solution to F.

The first call to the MCHS solver returns γ such that $\gamma(l_i) = \gamma(l_N) = 0$ for each $i = 1, .., n$. The assumption set is $A = \gamma$ and suppose the PB oracle returns a core $\kappa = \{l_1, l_N\}$. Without WCE, the IHS-based PBO algorithm subtracts $\overline{l_1}$ and $\overline{l_N}$ from A. Because there

Algorithm 3: IHS-based PBO solving algorithm with WCE.

```
Input  : A PB formula $F$ and a cost function O
Output: SAT/UNSAT, an optimal solution $\tau$
1  (sat?, $\kappa$, $\tau$) := PBSolve($F, \emptyset$);
2  if $sat?$ = UNSAT then
3      return (UNSAT, $\emptyset$);
4  $UB$ := O($\tau$), $\tau_{best} := \tau$, $K := \emptyset$;
5  while $TRUE$ do
6      $\gamma$ := MCHS(O, $K$);
7      $LB$ := O($\gamma$);
8      if $LB = UB$ then
9          return (SAT, $\tau_{best}$);
10     $W := \emptyset$;
11     for $(w, l) \in O$ do $W(l) := w$;            // Initialize working weight values
12     $A := \gamma$;
13     (sat?, $\kappa$, $\tau$) := PBSolve($F, A$);
14     while $sat?$ = UNSAT do
15         $w_{min} := \min_{l \in \kappa}\{W(l)\}$;                   // Obtain minimum weight
16         for $l \in \kappa$ do $W(l) := W(l) - w_{min}$;          // Subtract minimum weight
17         $K := K \cup \{\sum_{l \in \kappa} l \geq 1\}$;
18         $A := A \setminus \{\bar{l} \mid W(l) = 0\}$;        // Remove literals from A based on W
19         (sat?, $\kappa$, $\tau$) := PBSolve($F, A$);
20     if $O(\tau) < UB$ then
21         $UB$ := O($\tau$), $\tau_{best} := \tau$;
22     if $LB = UB$ then
23         return (SAT, $\tau_{best}$);
```

are no cores that do not include literal l_N, the disjoint phase terminates. Hence the algorithm without WCE extracts one core at a time for each disjoint phase.

Now consider the algorithm with WCE. Starting with $A = \{\overline{l_1}, \overline{l_2}, ..., \overline{l_n}, \overline{l_N}\}$, the working weights are set as $W(l_i) = 1$ for $i = 1, ..., n$ and $W(l_N) = n$. Suppose the PB oracle returns $\kappa = \{l_1, l_N\}$. Now $w_{min} = 1$ and working weights of the two literals of κ are $W(l_1) = 0$ and $W(l_N) = n - 1$. The literal $\overline{l_1}$ is removed from the assumption set, however, $\overline{l_N}$ still remains. Notice that $\{l_N\}$ is not a core, because an assignment that sets l_N to zero and all other literals to one is a solution to F. Therefore every core contains a literal with a working weight of value 1, which means that $W(l_N)$ is always subtracted by at most 1. This means that up to n cores may be extracted from a single disjoint phase. The disjoint phase ends when $W(l_N)$ is set to zero or the negations of all literals other than l_N are removed from A. $\triangle$

5.5 Seeding constraints to the MCHS solver

In the implementation of the PBO-IHS solver, an IP solver acts as an MCHS solver, solving the instances of the MCHS problem as IPs using the encoding presented in Section 2.4. When PBO-IHS is initialised, the IP consists of the cost function and an empty set of constraints. Providing additional information of the given PBO instance to the MCHS solver can cause it to return hitting sets that violate less PB constraints of the PBO instance. Several seeding techniques have been employed succesfully for IHS-based MaxSAT solving [29].

The seeding technique used in PBO-IHS is as follows. Before executing the main loop of the IHS-based PBO algorithm, every PB constraint $C \equiv \sum_i a_i l_i \geq b$ in the formula F for which every l_i is such that $l_i \in \text{LIT}(\text{O})$ is seeded, i.e., C is included into the IP as a constraint.

Example 5.5. Consider the PBO instance with $F = \{(l_1 + 2l_2 + 3l_3 \geq 3),\ (l_1 + l_3 \geq 1), (2l_3 + \overline{l_4} \geq 5)\}$ and $\text{O} = l_1 + 2l_3 + 3l_4$. The first constraint contains a literal $l_2 \notin \text{LIT}(\text{O})$, which is why the constraint is not seeded. The second constraint is seeded because it consists of literals $l_1, l_3 \in \text{LIT}(\text{O})$. The third constraint is seeded because it consists of literals $l_3, \overline{l_4} \in \text{LIT}(\text{O})$. After each constraint is checked, the MCHS instance is $\{(l_1 + l_3 \geq 1), (2l_3 + \overline{l_4} \geq 5)\}$ with the cost function $\text{O} = l_1 + 2l_3 + 3l_4$. $\triangle$

Seeding constraints does not affect the correctness of the IHS-based PBO algorithm, because seeding does not prevent the MCHS solver from returning a hitting set assignment

γ that is a restriction of some solution to F. If γ does not satisfy one of the constraints in F, then F under an assumption set γ is unsatisfiable because an assignment τ for which $\tau \restriction_{\text{LIT(O)}} = \gamma$ also does not satisfy the same constraint. Seeding constraints to the MCHS instance also decreases the number of cores that may be returned by the PB oracle, which is demonstrated by the following example.

Example 5.6. Consider a PBO instance with $F = \{(\sum_{i=1}^{7} l_i^O \geq 3), (\sum_{i=1}^{7} l_i \geq 5), (\overline{l_1} + l_1^O \geq 1), (\overline{l_2} + l_2^O \geq 1), \ldots, (\overline{l_7} + l_7^O \geq 1)\}$ and $\text{O} = \sum_{i=1}^{7} l_i^O$. This instance contains 7 pairs of variables (l_i, l_i^O), for which $l_i \notin \text{LIT}(\text{O})$ and $l_i^O \in \text{LIT}(\text{O})$. Constraints of form $(\overline{l_i} + l_i^O \geq 1)$ establish an implication $l_i \Rightarrow l_i^O$. Due to the constraint $(\sum_{i=1}^{7} l_i \geq 5)$ in conjunction with $l_i \Rightarrow l_i^O$ constraints, the cost of a solution is restricted to have a cost of at least 5, implying that the condition $(\sum_{i=1}^{7} l_i^O \geq 5)$ must hold true.

Any subset $\kappa \subset \{l_i^O \mid i \in \{1,..,7\}\}$ for which $|\kappa| \geq 3$ is a core of F, because an assignment that sets at least 3 literals l_i^O to 0 is not a solution to F due to the implied $(\sum_{i=1}^{7} l_i^O \geq 5)$. There are $\sum_{i=3}^{7} \binom{7}{i}$ possible cores that the PB oracle may return.

If constraints are seeded, the constraint $(\sum_{i=1}^{7} l_i^O \geq 3)$ is included in the MCHS instance from the start of the execution of PBO-IHS. This means that a hitting set returned by the MCHS solver sets at most four literals l_i^O to 0. Therefore the PB oracle extracts a core κ for which $3 \leq |\kappa| \leq 4$, which means that there are at most $\binom{7}{3} + \binom{7}{4}$ cores that the PB oracle will extract. △

5.6 Non-optimal hitting sets

Using an upper bound on the cost of an optimal solution *UB*, PBO-IHS can be implemented with an MCHS solver that returns non-optimal hitting sets. The requirement of the non-optimal MCHS solver is that it returns a hitting set γ that is either minimum-cost or for which $\text{O}(\gamma) < UB$. The non-optimal MCHS solver should also be able to explicitly identify if the returned γ is non-optimal.

If a hitting set assignment γ returned by the non-optimal MCHS solver is not minimum-cost, then it cannot be used to compute the lower bound (LB). For a non-optimal hitting set assignment γ, there might be γ' such that $\text{O}(\gamma') < \text{O}(\gamma)$ for which there exists a solution τ to F such that $\tau \restriction_{\text{LIT(O)}} = \gamma'$, which means $\text{O}(\tau) = \text{O}(\gamma') < \text{O}(\gamma)$. The value of *LB* is only updated to $\text{O}(\gamma)$ when the non-optimal MCHS solver identifies the returned γ to be minimum-cost.

It is possible that F is satisfiable under an assumption set γ, where γ is a non-optimal hitting set. The PB oracle returns τ such that $\text{O}(\tau) = \text{O}(\gamma) < UB$, which causes the UB to be updated to $\text{O}(\gamma)$. The IHS-based PBO algorithm does not terminate if $LB < \text{O}(\gamma) = UB$. In that case the algorithm proceeds to the next iteration without extracting any cores. However, because UB decreases in such scenario, the algorithm will still eventually terminate. The correctness of the IHS-based MaxSAT algorithm that uses the non-optimal MCHS solver is proven in [10]. The proof can be adapted to IHS-based PBO algorithm as well.

Example 5.7. Consider a PBO instance with $F = \{\sum_{i=1}^{5} l_i \geq 3\}$ and $\text{O} = \sum_{i=1}^{5} l_i$. We clearly see that the cost of an optimal solution is 3. Suppose PB oracle returns τ for which $\tau(l_i) = 1$ for each $i = 1, 2, .., 5$, setting $UB = \text{O}(\tau) = 5$. Suppose then that the first call to the MCHS solver returns γ for which $\gamma(l_1) = 0$ and $\gamma(l_i) = 1$ for each $i = 2, .., 5$ because $\text{O}(\gamma) = 4 < UB$. The PB oracle is called under the assumption set γ, which returns `SAT` and a solution τ for which $\tau(l_1) = 0$ and $\tau(l_i) = 1$ for $i = 2, .., 5$. The upper bound UB is updated to $\text{O}(\tau) = 4$. Then the MCHS solver is called again, and suppose it returns γ for which $\gamma(l_1) = \gamma(l_2) = 0$ and $\gamma(l_i) = 1$ for each $i = 3, .., 5$ because $\text{O}(\gamma) = 3 < UB$. The PB oracle called under γ returns `SAT` with solution τ such that $\tau(l_1) = \tau(l_2) = 0$ and $\tau(l_i) = 1$ for $i = 3, 4, 5$. The upper bound UB is updated to $\text{O}(\tau) = 3$, which is the cost of an optimal solution to F. From this point forward if returned γ is non-optimal, then $\text{O}(\gamma) < 3$, which means that the PB oracle will return `UNSAT` because there are no solutions τ for which $\text{O}(\tau) < 3$. A hitting set returned by the MCHS solver will always cause the PB oracle to extract a core, until enough cores are extracted for the MCHS solver to return γ with a cost of 3. Then LB is updated to the value of 3, triggering the halting condition $LB = UB$. △

When solving for a non-optimal hitting set, the hitting set solver may avoid the worst case exponential running time that would be required when solving for a minimum-cost hitting set. In the implementation of PBO-IHS we make use of an IP solver that maintains the best solution candidate internally when solving the MCHS instance as an IP. The IP solver is interrupted when an internal solution candidate is found for which the cost is strictly lower than the most recent UB value. In addition, with each MCHS call a flag is returned that is set to true if the IP solver was not interrupted, which means that the IP solver has executed the solving algorithm to its completion, producing an optimal solution to the MCHS instance.

An alternative strategy can be used in implementing the IHS-based PBO algorithm with a non-optimal MCHS solver. Suppose the PB oracle returns `SAT` with a solution τ without extracting any cores. Instead of solving for another non-optimal hitting set whose cost is strictly lower than $\mathrm{O}(\tau)$, the MCHS solver is forced to find a minimum-cost hitting set γ. This forces LB to be updated to $\mathrm{O}(\gamma)$. After forcing the MCHS solver to return a minimum-cost hitting set, if the PB oracle returns `SAT` with solution τ', then $\mathrm{O}(\tau') = \mathrm{O}(\gamma) = LB$, terminating the algorithm. In the other case where the PB oracle returns `UNSAT`, the cores are extracted as usual, after which the MCHS solver is called without the requirement of the returned γ to be optimal. The IHS-based PBO algorithm using this strategy is still correct because the requirement of the non-optimal MCHS solver is still the same – it either returns an optimal hitting set or a hitting set with a cost strictly less than UB.

Example 5.8. Consider the same PBO instance as in Example 5.7, with UB set initially to 5. Suppose the first call to the MCHS solver returns γ for which $\gamma(l_1) = 0$ and $\gamma(l_i) = 1$ for $i = 2, .., 5$ because $\mathrm{O}(\gamma) = 4 < UB$. The PB oracle returns `SAT` and UB is updated to 4. Because the PB oracle returned `SAT` without extracting any cores, the next call to the MCHS solver forces it to return an optimal hitting set. As the MCHS instance has no constraints, the hitting set is γ for which $\gamma(l_i) = 0$ for $i = 1, .., 5$. The PB oracle returns `UNSAT` and suppose $\kappa = \{l_1, l_2, l_3\}$.

After this, the MCHS solver is called to find a non-optimal hitting set for the constraint set $\{(l_1 + l_2 + l_3 \geq 1)\}$, where the cost of the hitting set is strictly less than $UB = 4$. Suppose the MCHS solver returns γ for which $\gamma(l_1) = \gamma(l_2) = 0$ and $\gamma(l_i) = 1$ for $i = 3, 4, 5$. The PB oracle returns `SAT` and UB is updated to 3. The MCHS solver is again forced to solve for an optimal hitting set, suppose it returns $\gamma(l_1) = 1$ and $\gamma(l_i) = 0$ for $i = 2, .., 5$, from which the PB oracle returns `UNSAT` with $\kappa = \{l_2, l_3, l_4\}$. After UB is set to 3, the PB oracle cannot return `SAT` when a non-optimal hitting set is used as an assumption set, because there are no solutions to F with cost less than 3. Therefore the progression for the rest of the algorithm is as described in Example 5.7. $\triangle$

5.7 Fixing individual variables

Reduced cost fixing is a standard technique used in IP solving [26, 28, 77] and has found its footing in IHS-based MaxSAT solving [10]. With each MCHS call, the IP that encodes the MCHS instance is solved by an IP solver, which also retrieves an optimal solution $\tau_{LP} : \text{VAR}(\mathrm{O}) \to [0,1]$ to the LP relaxation of the IP, its cost $z_{LP} = \mathrm{O}(\tau_{LP})$, and the reduced cost $d_i \in \mathbb{R}$ for each literal $l_i \in \text{LIT}(\mathrm{O})$ that has an integral assignment, i.e., $\tau_{LP}(l_i) \in \{0,1\}$. The reduced cost d_i of a literal l_i is the lower bound for the increase in the cost function if the assignment of l_i is changed to its opposite integer value. Because z_{LP} is optimal, the cost can only increase if the assigned value to l_i is changed to its opposite, increasing the cost of the solution at minimum by the value of d_i.

When fixing a variable, for example l_i to 0, we need to make sure that there exists a solution τ to F such that $\tau(l_i) = 0$ and $\mathrm{O}(\tau) \leq \mathrm{O}(\tau')$ for any solution τ' to F such that $\tau'(l_i) = 1$. The PBO-IHS solver maintains the best solution candidate τ_{best} and its cost as the upper bound on the cost of an optimal solution, $UB = \mathrm{O}(\tau_{best})$. Suppose that $\tau_{LP}(l_i) = 0$ and $z_{LP} + d_i > UB$. This means that a solution to the LP relaxation τ'_{LP} such that $\tau'_{LP}(l_i) = 1$ has a cost $\mathrm{O}(\tau'_{LP}) > UB$. Recall from Section 2.2 that the cost of an optimal solution to the IP is at least the cost of an optimum solution to its LP relaxation. Therefore $\mathrm{O}(\tau') \geq \mathrm{O}(\tau'_{LP}) > UB$ for any solution τ' to F such that $\tau'(l_i) = 1$, which means that $\tau_{best}(l_i) = 0$. Therefore it is safe to fix l_i to 0. Using similar reasoning, l_i can be fixed to 1 if $\tau_{LP}(l_i) = 1$ and $z_{LP} - d_i > UB$, keeping in mind that $d_i \leq 0$ if $\tau_{LP}(l_i) = 1$.

In the case where the absolute value of d_i added on top of z_{LP} is equal to UB, the condition $\tau_{best}(l_i) = \tau_{LP}(l_i)$ needs to be explicitly checked, because otherwise there is no guarantee that there is a solution τ to F such that $\tau(l_i) = \tau_{LP}(l_i)$ and $\mathrm{O}(\tau) \leq UB$. In summary, the conditions for fixing l_i to a specific value are as follows.

$$\tau_{LP}(l_i) = 0 \text{ and } \begin{cases} (z_{LP} + d_i > UB) \text{ or} \\ (z_{LP} + d_i = UB \text{ and } \tau_{best}(l_i) = 0) \end{cases} \to \text{ fix } l_i \text{ to } 0$$

$$\tau_{LP}(l_i) = 1 \text{ and } \begin{cases} (z_{LP} - d_i > UB) \text{ or} \\ (z_{LP} - d_i = UB \text{ and } \tau_{best}(l_i) = 1) \end{cases} \to \text{ fix } l_i \text{ to } 1$$

Reduced cost fixings can also be derived from the LP relaxation of the entire PBO instance. As the PBO instance remains unchanged throughout the execution of PBO-IHS, the LP relaxation needs only to be solved once to retrieve the cost of an optimal solution to the

LP relaxation and a reduced cost for each literal $l_i \in \textsc{var}(F)$. The conditions for fixing individual assignments to l_i are the same as detailed previously and need to be rechecked only when UB is decreased due to a changed τ_{best}.

Another variable fixing technique used in PBO-IHS is employed when the PB oracle extracts a unit core κ such that $|\kappa| = 1$. If $\kappa = \{l\}$ is a core of F, then for every solution τ to F, $\tau(l) = 1$. When such core is extracted, the constraint $(l \geq 1)$ is added to the MCHS instance, effectively fixing the literal l. With unit core fixing, the same literal l is explicitly fixed in the PB oracle as well.

6 Experiments

In this section we present results from an empirical evaluation of the PBO-IHS solver. In Section 6.1 we detail the implementation of the PBO-IHS solver, which was used along with other PBO solvers to solve benchmarks that were collected and sampled as described in Section 6.2. We discuss the results of the experiments in Section 6.3. The experiments were run single-threadedly on nodes with 8-core Intel Xeon E5-2670 2.6-GHz CPUs and 64-GB RAM with a per-instance 3600-second time and 16-GB memory limit.

6.1 Implementation

We implemented the PBO-IHS solver in Python (version 3.6.9), with a PB solver (acting as the PB oracle) and an IP solver (acting as the MCHS solver) imported as external modules. We use Roundingsat version 2 [41] (commit 1476bf0bcd) as the PB solver. The Roundingsat solver was compiled with a configuration that enables learning constraints from Farkas multipliers [32]. The Roundingsat code published in [40] does not support solving a PB formula under an assumption set and cannot extract cores. To make the PB solver compatible with PBO-IHS, we extended the Roundingsat implementation to be able to extract cores by including an analyzeFinal function similar to the one implemented in MiniSat SAT solver [37, 38]. As the IP solver we use IBM ILOG CPLEX C++ API version 12.8 [25]. As both Roundingsat and the CPLEX API are implemented in C++, we use pybind11 to compile the C++ libraries as python modules. A benefit of using both solvers as external modules is that solvers acting as the PB oracle or the MCHS solver can be replaced with other solvers with relative ease in future research.

We make sure that the instantiated solvers persist throughout the execution of PBO-IHS, rather than reinstantiating the solvers with each solver call. When a core is added to the MCHS solver, PBO-IHS performs a call to CPLEX to add another constraint, modifying the IP instance. The instance in Roundingsat is modified when PBO-IHS fixes a variable as detailed in Section 5.7. As modifications to the two solvers are limited to adding a constraint on top of an existing set of constraints, the internal state of either solver does not get invalidated in-between solver calls.

Unless otherwise stated, in the following sections PBO-IHS refers to our implementation of the IHS-based PBO algorithm with the following default configuration.

- Subset minimization is not enabled (recall Section 5.1).
- For each call to the PB oracle, the assumption set is shuffled 20 times, with only the smallest core being included in the MCHS instance (recall Section 5.2).
- Disjoint phase is included using weight-aware core extraction (recall Sections 5.3 and 5.4).
- MCHS seeding is enabled (recall Section 5.5).
- The MCHS solver is configured to return non-optimal hitting sets, but it is forced to return an optimal hitting set if the previous hitting set caused no cores to be extracted by the PB solver (recall Section 5.6).
- Reduced cost fixing is enabled based on the LP relaxation of the MCHS instance (recall Section 5.7).
- Reduced cost fixing is not enabled based on the LP relaxation of the entire PBO instance (recall Section 5.7).
- Unit cores are fixed in the PB solver (recall Section 5.7).

The PBO-IHS implementation is available in open source at

`https://bitbucket.org/coreo-group/pbo-ihs-solver/`.

6.2 Benchmarks

We collected a large number of benchmarks from two sources:

- the Pseudo-Boolean Competition 2016 website [82] (which also contains benchmarks from the previous instantiations of the PB competition since 2005), and
- the MIPLIB 2017 library [50] (which includes benchmarks from earlier MIPLIB releases).

Most benchmarks collected from these sources were not suitable for our experiments for different reasons. The set of 17312 Pseudo-Boolean Competition benchmarks and 1273 MIPLIB benchmarks contained 9462 satisfiable PBO benchmarks. 548 benchmarks were removed because they contain a non-linear term either in the cost function or in one of the constraints. 206 benchmarks were removed because they have at least one coefficient with an absolute value higher than 2^{64}. In total, we were left with 8708 benchmarks that were used for our experiments.

We found that the benchmark set is significantly unbalanced with respect to the number of benchmarks per each problem domain. For a fair comparison of the overall performance of the different solvers across the different benchmark domains, we sampled at random (without repetition) from each problem domain up to 30 instances. The sampled benchmark set contains in total 1786 benchmarks. Results reported in the following section are with respect to the sampled benchmark set by default, and we refer to the set of 8708 benchmarks as the *full* benchmark set. We categorized the benchmarks to the best of our knowledge into different problem domains based on their source, related publications and by their file names. The problem domains with citations and brief descriptions are detailed in Appendix A.

6.3 Results

We make several observations from the results of our empirical evaluation. We compare the performance of PBO-IHS with other published specialized PBO solvers in Section 6.3.1 and find that PBO-IHS performs best with respect to the sampled benchmark set. We analyze the effect of different configurations of PBO-IHS in Section 6.3.2. We then compare the performance of PBO-IHS with a commercial MIP solver CPLEX in Section 6.3.3. Finally, in Section 6.3.4 we show the division of PBO-IHS solving time between the MCHS solver and the PB oracle.

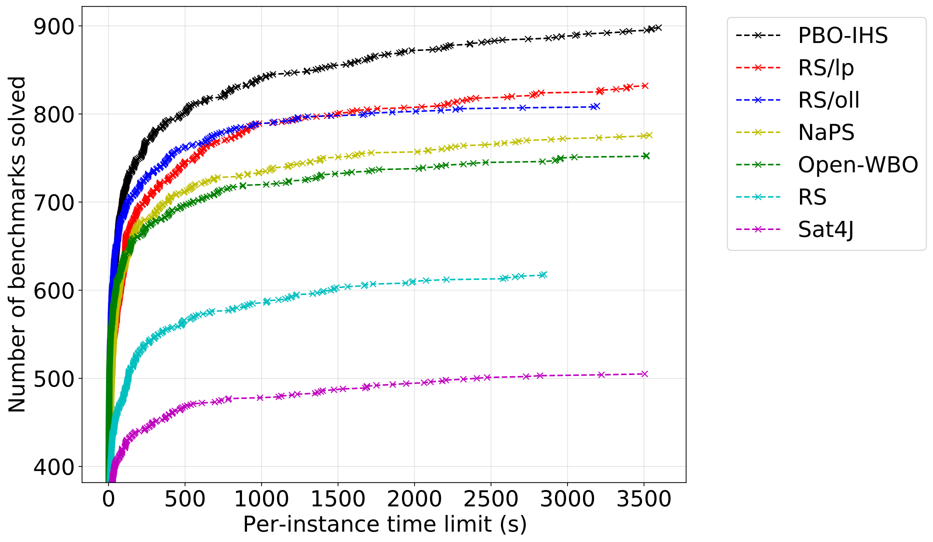
6.3.1 Comparison with specialized PBO solvers

The empirical performance of PBO-IHS is compared to the performance of other specialized PBO solvers on the sampled benchmark set. Following PBO solvers were used for the comparison.

- **RS** – Roundingsat solver [41] (commit 1476bf0bcd) with solution-improving search.
- **RS/lp** – Roundingsat solver (commit 1476bf0bcd) with solution-improving search and with the use of Farkas multipliers [32] for constraint learning.
- **RS/oll** – Roundingsat solver that extends the OLL algorithm to PBO solving as described in [33].
- **Sat4J** – PB solver with solution-improving search [67].
- **Open-WBO** – OLL-based MaxSAT solver that encodes PB constraints to CNF clauses using generalized totalizer encoding [73].
- **NaPS** – SAT solver that encodes PB constraints to CNF clauses using binary decision diagrams [90]. Optimization is done with a combination of solution-improving search and binary search.

Figure 6.1 (top) shows how many benchmarks each solver was able to solve (y-axis) under different per-instance time limits (x-axis). We observe that PBO-IHS outperforms all of the other specialized solvers. The two recent variants of Roundingsat perform the second and third best. In particular, PBO-IHS also outperforms the version of Roundingsat (RS/lp) which is used within PBO-IHS for core extraction. In Figure 6.1 (bottom) the results of the three best-performing solvers are shown under 10 different random samplings of the sampled benchmark set. For each solver S, Figure 6.1 (bottom) includes 3 lines: S-max, S-median and S-min. A point (t, x) on the S-max line indicates that S was able to solve x benchmarks within t seconds for *at least one* of the ten benchmark set samples. Analogously, a point on the S-min line indicates solving x benchmarks within t seconds in *all* samples, and the S-median line indicates solving x benchmarks within t in *five* of the 10 samples. This indicates that the ranking of the three best-performing solvers is robust with respect to the different random samplings of the sampled benchmark set.

More detailed data per benchmark domain over the *full* benchmark set is reported in Table 6.1, with the number of instances solved (left column) and the cumulative runtimes over solved instances (right column) shown for each solver. In Figure 6.2, PBO-IHS is compared to RS/lp and RS/oll, where for each benchmark in the sampled benchmark set a tick is placed based on the PBO-IHS solve time on the x-axis and based on either Roundingsat version solve time on the y-axis, with the color of the tick differentiating the problem domain of the benchmark. It can be observed that the relative performance of the Roundingsat versions and PBO-IHS depends significantly on the problem domain, suggesting that the approaches complement each other.



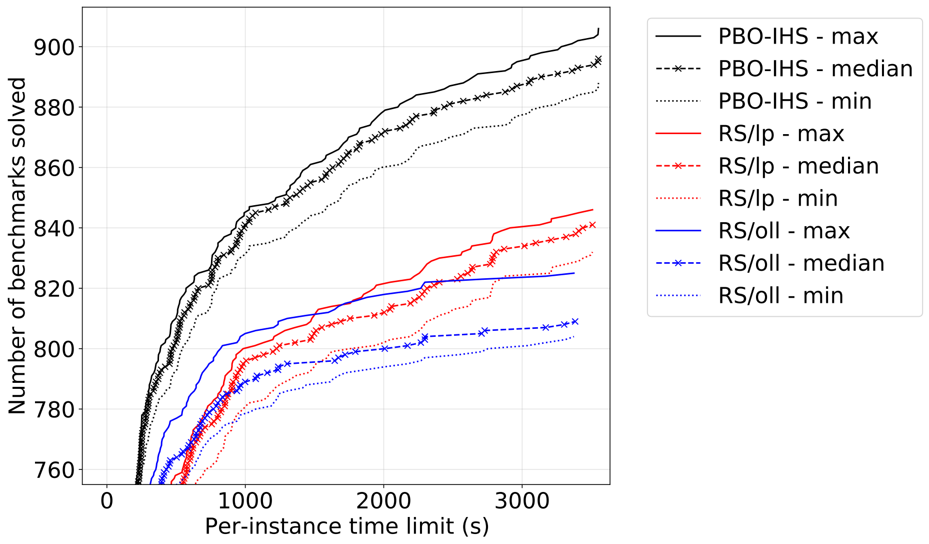


Figure 6.1: Top: Runtime comparison of specialized PBO solvers. Bottom: Confidence intervals over 10 benchmark subset samples for the three best-performing solvers.

Table 6.1: Comparison of specialized PBO solver per benchmark domain: number of solved instances (#) and cumulative runtimes over solved instances in seconds (cum.)

	Sat4J		RS		Open-WBO		Naps		RS/lp		RS/oll		PBO-IHS	
Domain (#instances)	#	cum.	#	cum.	#	cum.	#	cum.	#	cum.	#	cum.	#	cum.
10orplus/9orless (156)	55	99459	39	64252	156	**202**	156	14149	154	55344	156	1406	156	23670
caixa (24)	24	13	20	16	24	**2**	24	179	24	70	24	3	24	64
rand.*list (118)	113	5241	59	1961	118	**44**	118	2218	118	692	118	125	118	2296
area_* (59)	11	626	37	11998	**59**	138	54	3613	54	16176	57	9469	51	11784
trarea_ac (18)	1	1	1	2	13	2314	4	4582	16	3722	5	1868	**18**	7751
aries-da_nrp (70)	15	1747	16	7994	25	11938	19	7325	**43**	15442	21	10599	32	10413
BA (1440)	85	175161	301	221066	160	116377	0	0	**588**	472938	356	230143	20	30038
NG (960)	2	804	59	71042	11	11990	0	0	48	115499	**138**	194128	0	0
MANETs (150)	29	5744	0	0	20	13648	14	17875	**40**	23547	29	9525	25	21152
BioRepair (30)	30	457	30	8551	30	105	30	311	30	3258	30	**35**	30	262
Metro (30)	30	4413	30	1270	30	3341	30	**775**	30	1795	29	3291	27	12595
ShiftDesign (30)	12	2258	16	5671	28	10696	**30**	2781	18	12824	27	3371	9	9060
Timetabling (30)	17	11920	15	8026	27	10054	25	17502	23	15419	24	3295	**28**	8768
EmployeeScheduling (14)	0	0	0	0	9	**480**	9	506	0	0	0	0	0	0
golomb-rulers (34)	14	**642**	14	5765	11	1656	12	3451	12	1216	12	436	12	4212
bsg (60)	0	0	10	**156**	10	4767	10	813	10	465	10	1963	5	16
mis/mds (120)	0	0	44	8968	48	6605	47	6245	45	3853	45	5525	**57**	15335
course-ass (6)	0	0	2	1225	2	29	**4**	3226	3	33	2	1	1	6
decomp (10)	0	0	0	0	8	**1809**	8	4516	0	0	2	2200	0	0
data (68)	1	2	8	1628	0	0	4	2414	13	**4044**	13	5837	11	2163
dt-problems (60)	37	1712	40	3573	38	2777	59	8697	60	**2**	60	7	60	113
domset (15)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
factor (186)	186	56	186	**0**	186	710	186	160	186	2	186	**0**	186	342
factor-mod-B (225)	0	0	225	67	199	39899	225	3243	225	60	225	**25**	225	344
fctp (35)	2	36	2	0	1	141	6	468	5	940	5	2	**12**	499
featureSubscription (20)	20	1266	20	2492	20	**76**	20	112	20	8106	20	941	20	303
frbXX-XX-opb (40)	0	0	0	0	0	0	**17**	11552	0	0	0	0	6	11343
flexray (9)	**5**	1697	4	83	4	496	4	296	4	393	4	31	4	50
fome (3)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
graca (100)	20	230	24	3664	97	4687	**98**	10487	31	21769	93	14428	84	40593
haplotype (8)	0	0	8	202	8	**31**	8	60	8	2385	8	57	7	4023
garden (7)	4	28	5	1	6	94	6	355	5	1	5	0	6	**76**
hw32/hw64/hw128 (27)	0	0	2	1	1	217	1	50	8	3470	5	889	**10**	12063
jXXopt (2040)	1604	32395	1613	36840	1611	34453	**1621**	58870	1589	51821	1603	42149	1579	64191
keeloq_tasca (4)	0	0	4	424	4	360	4	3019	4	33	4	**7**	4	54
kullmann (7)	0	0	1	2	1	0	0	0	1	2	1	3	**3**	3016
lion9-single-obj (1513)	1181	89655	687	4242	**1501**	12026	1400	105853	1412	113829	1482	62955	1487	120526
logic-synthesis (74)	24	7180	39	5078	49	4750	33	2581	61	11647	48	786	**71**	708
miplib/neos (79)	18	3516	27	1725	25	2455	25	5479	37	8377	32	6048	**38**	10631
miplib/other (405)	84	9044	96	7093	80	20989	95	20135	147	36264	123	15349	**156**	38501
unibo (36)	0	0	3	127	0	0	0	0	3	228	3	77	**8**	5342
market-split (20)	2	659	4	4750	0	0	4	1575	4	**342**	4	2670	1	1167
opb/graphpart (31)	0	0	8	2641	22	940	23	7019	12	435	14	4942	**24**	5211
opb/autocorr_bern (43)	0	0	5	1168	3	1768	3	337	4	3594	3	318	**8**	2089
opb/sporttournament (22)	0	0	4	667	7	697	4	168	4	23	6	2032	**11**	3121
opb/edgecross (19)	0	0	3	2869	6	1634	4	1230	6	2899	3	9	**12**	3984
opb/pb (8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
opb/faclay (10)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	960
opb/other (6)	0	0	1	**0**	1	**0**	1	**0**	1	**0**	1	**0**	1	2
primes/aim (48)	48	15	48	**0**	48	**0**	48	**0**	48	4	48	**0**	46	234
primes/jnh (16)	16	16	16	35	16	36	16	**11**	16	19	16	44	16	53
primes/ii (41)	10	504	21	7087	26	9348	25	12121	23	6874	33	2792	**34**	5230
primes/par (30)	20	17	20	14	20	**2**	20	14	20	15	20	31	20	422
primes/other (13)	2	5	2	2	6	**5**	6	22	6	452	4	204	5	938
routing (15)	15	1030	15	19	15	2	15	17	15	7	15	**1**	15	26
radar (12)	0	0	6	313	0	0	0	0	6	71	1	127	**12**	77
synthesis-ptl-cmos (10)	2	0	2	0	8	15	3	27	9	135	8	1186	**10**	16
testset (6)	6	1529	6	1161	5	81	6	1721	6	**0**	6	1	6	8
ttp (8)	2	1	2	0	2	**0**	2	1	2	0	2	**0**	2	10
vtxcov (15)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
wnq (15)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

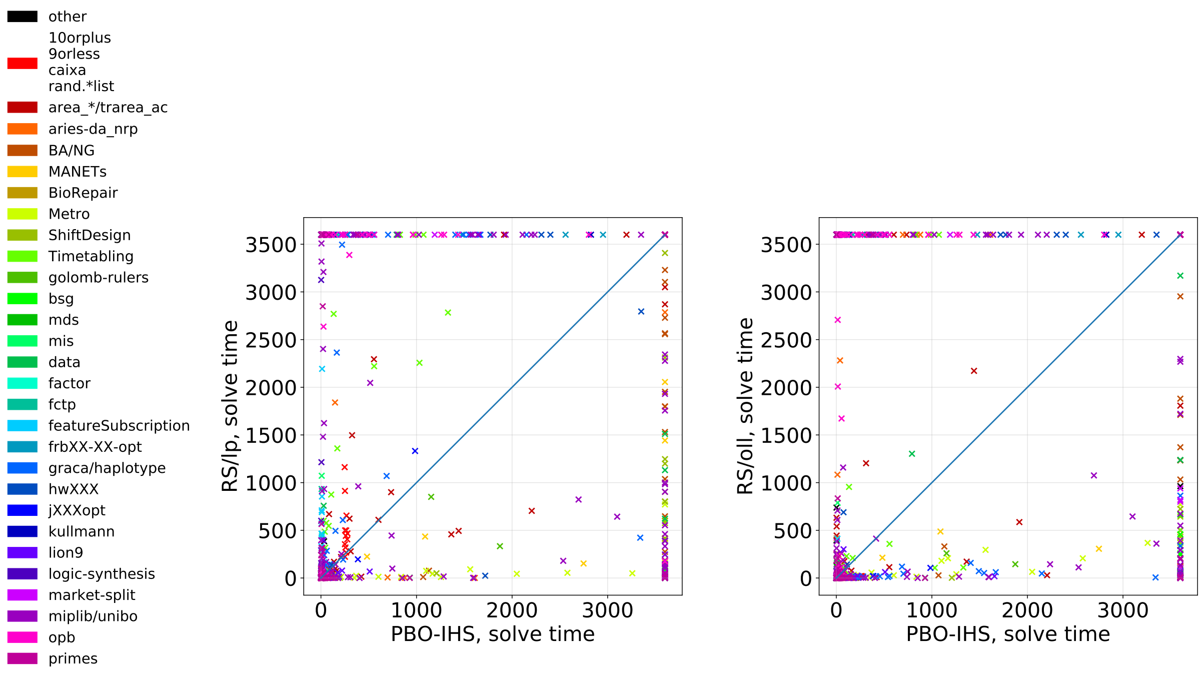


Figure 6.2: Per-instance runtime comparison between PBO-IHS (x-axis) and (i) RS/lp (y-axis, left), (ii) RS/oll (y-axis, right).

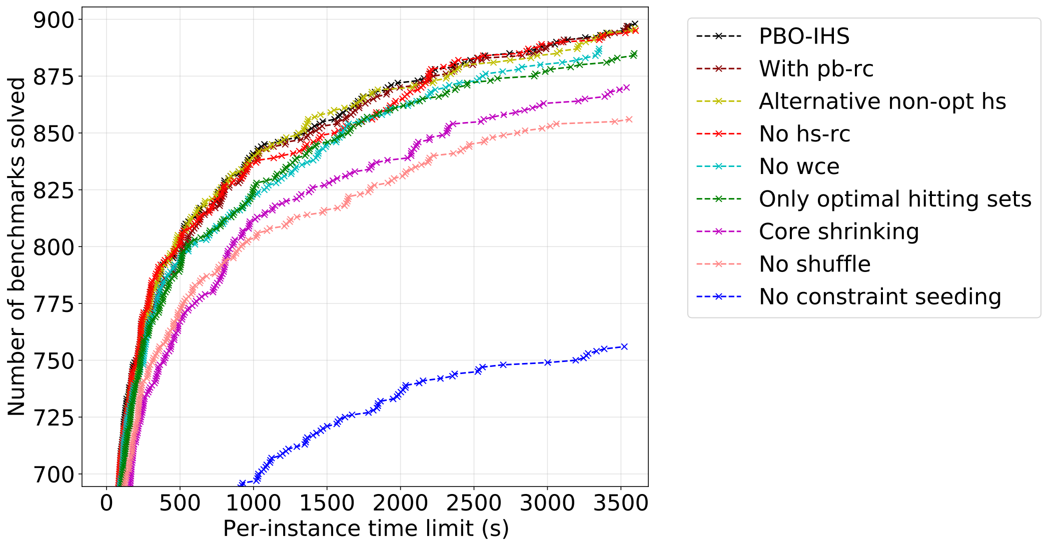


Figure 6.3: Runtime comparison of various PBO-IHS variants.

6.3.2 Impact of different search techniques in PBO-IHS

The marginal impact of the different search techniques and refinements on the empirical performance of PBO-IHS is reported in Figure 6.3, which provides a comparison of the default configuration of PBO-IHS as described in Section 6.1 to configurations that differ from the default configuration in the following ways.

- **With pb-rc** – reduced cost fixing is enabled based on the LP relaxation of the entire PBO instance as described in Section 5.7.
- **Alternative non-opt hs** – the MCHS solver is never forced to find an optimal hitting set, one of the strategies described in Section 5.6.
- **No hs-rc** – reduced cost fixing is disabled based on the LP relaxation of the MCHS instance as described in Section 5.7.
- **No wce** – the disjoint phase is implemented as described in Section 5.3 rather than as described in Section 5.4.
- **Only optimal hitting sets** – the MCHS solver always returns an optimal hitting set rather than as described in Section 5.6.
- **Core shrinking** – each extracted core is shrinked using a subset-minimization algorithm as described in Section 5.1.
- **No shuffle** – the assumption set is not shuffled, with only one core getting extracted per call to the PB oracle rather than as described in Section 5.2.
- **No constraint seeding** – no PB constraints are seeded to the MCHS solver rather than as described in Section 5.5.

We observe that constraint seeding makes the largest positive marginal contribution to the empirical performance of PBO-IHS, and assumption set shuffling the second largest positive marginal contribution. The third largest positive contribution is made by using non-optimal hitting sets, followed closely by weight-aware core extraction. Performing core shrinking impacts negatively on the empirical performance of PBO-IHS, ranking as the third worst PBO-IHS configuration out of nine configurations. The two different forms of reduced cost fixing have only a very modest impact. While reduced cost fixing based on the LP relaxation of the entire PBO instance does not make a significant negative marginal contribution, it does not appear to improve on the performance of PBO-IHS, which justifies disabling it in the default configuration of PBO-IHS. The alternative strategy for using the non-optimal MCHS solver does not appear to alter the performance of PBO-IHS in any significant capacity.

Recall from Section 5.2 that when a set of cores is produced by shuffling the assumption set, there are several strategies that can be used to decide which cores are included from the core set. The effect of different strategies is reported in Figure 6.4 with the following strategies.

- **No shuffle** – the assumption set is not shuffled with only one core getting extracted per call to the PB oracle.
- **Shuffle once** – the assumption set is permuted at random before calling the PB oracle, extracting one core.
- **20 shuffles 1 core** – for each call to the PB oracle, the assumption set is shuffled 20 times, with only the smallest core being included in the MCHS instance.
- **20 shuffles 5 cores** – for each call to the PB oracle, the assumption set is shuffled 20 times, with five smallest cores being included in the MCHS instance.
- **20 shuffles 20 cores** – for each call to the PB oracle, the assumption set is shuffled 20 times, with all cores being included in the MCHS instance.

For each assumption set shuffling strategy we also ran a PBO-IHS variant that did not use WCE during the disjoint phase.

We observe that regardless of the assumption set shuffling strategy, using WCE during the disjoint phase always improves on the performance of PBO-IHS. The ranking of the shuffling strategies with respect to the number of benchmarks solved within the 3600 second time limit is the same among variants that use WCE and among variants that do not. The *no shuffle* strategy is the weakest performing shuffling strategy from the strategy set. The *shuffle once* strategy differs from the *no shuffle* strategy only in that the assumption set is permuted once before calling the PB oracle, which is enough to improve the performance of PBO-IHS in a significant way. Including all cores from 20 shuffles does not improve the performance of PBO-IHS significantly when compared to the *shuffle once* strategy. The *20 shuffles 5 cores* strategy is the second best strategy between the five strategies and the WCE variant is quite competitive with the default configuration of PBO-IHS.

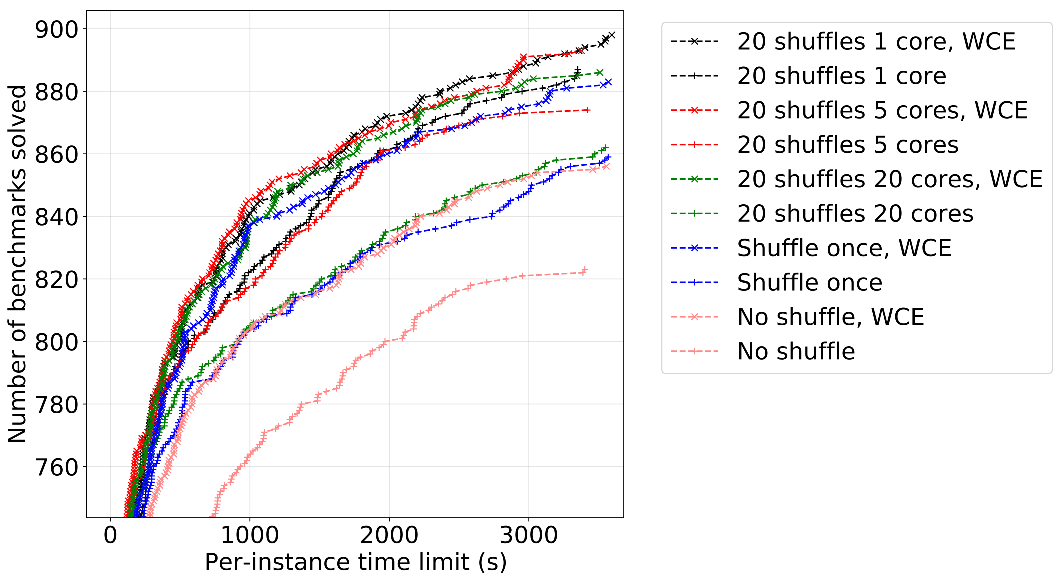


Figure 6.4: Runtime comparison of PBO-IHS variants that differ in assumption set shuffling strategies. For each strategy, a variant is included that uses WCE during the disjoint phase and a variant that does not.

6.3.3 Comparison with a commercial IP solver

The runtime performance of PBO-IHS is compared separately against CPLEX [25], one of the state-of-the-art commercial MIP solvers with a significant number of person years behind it. For a fair comparison with CPLEX, the CPLEX presolver was used before running PBO-IHS as well. This eliminates to an extent the differentiating contribution of the powerful preprocessor of CPLEX in terms of runtime performance, though it should be noted that CPLEX appears to employ further probing for e.g. clique inequalities after the presolving stage, which we were unable to employ before running PBO-IHS. A per-instance runtime comparison for the sampled benchmark set is shown in Figure 6.5, with more details per benchmark domain provided in Table 6.2 on the *full* benchmark set. We observe that, while CPLEX fairs better in the overall number of solved instances, the two solvers exhibit noticeably complementary performance depending on the problem domain considered.

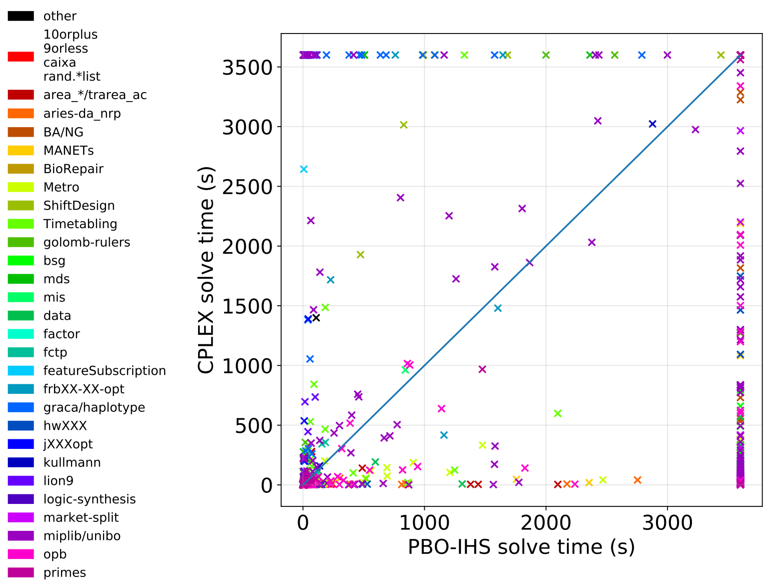


Figure 6.5: Per-instance runtime comparison of PBO-IHS (x-axis) vs CPLEX (y-axis).

Table 6.2: Per-domain comparison of PBO-IHS and CPLEX: number of solved instances (#) and cumulative runtimes over solved instances in seconds (cum.)

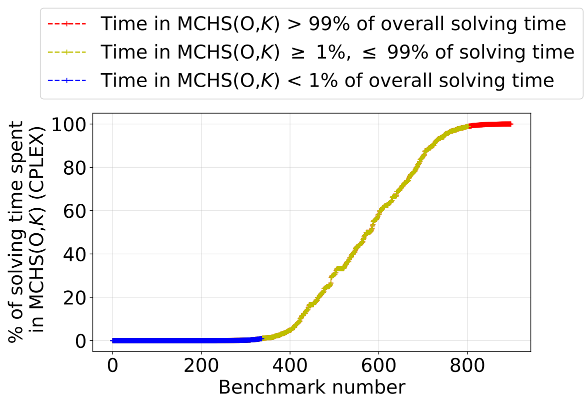
	PBO-IHS		CPLEX	
Domain (#instances)	#	cum.	#	cum.
10orplus/9orless (156)	156	20309	156	**1709**
caixa (24)	18	38	**24**	61
rand.*list (118)	118	878	118	**301**
area_* (59)	57	10735	**59**	789
trarea_ac (18)	17	5209	**18**	47
aries-da_nrp (70)	55	17508	**70**	2278
BA (1440)	7	17028	**761**	419659
NG (960)	0	0	**238**	224058
MANETs (150)	27	13051	**61**	25757
BioRepair (30)	30	**223**	30	862
Metro (30)	27	10911	**30**	2626
ShiftDesign (30)	**10**	9688	6	7062
Timetabling (30)	**28**	8019	27	6313
EmployeeScheduling (14)	0	0	**13**	149
golomb-rulers (34)	**12**	4669	10	589
bsg (60)	5	16	**15**	3571
mis/mds (120)	**64**	26665	58	15127
course-ass (6)	1	8	**6**	12
decomp (10)	0	0	0	0
data (68)	10	2202	**24**	3076
dt-problems (60)	47	74	**60**	358
domset (15)	0	0	0	0
factor (186)	186	348	186	**242**
factor-mod-B (225)	**225**	317	216	4204
fctp (35)	12	**622**	12	936
featureSubscription (20)	**20**	301	1	2644
frbXX-XX-opb (40)	**5**	5397	3	3615
flexray (9)	**4**	69	3	14
fome (3)	0	0	0	0
graca (100)	**62**	20019	27	14459

	PBO-IHS		CPLEX	
Domain (#instances)	#	cum.	#	cum.
haplotype (8)	**7**	2992	0	0
garden (7)	6	76	6	**60**
hw32/hw64/hw128 (27)	6	1324	**18**	5072
jXXopt (2040)	**1581**	47081	1487	136243
keeloq_tasca (4)	4	**124**	4	1412
kullmann (7)	3	**3016**	3	3183
lion9-single-obj (1513)	**1487**	33403	1480	57923
logic-synthesis (74)	71	767	71	**690**
miplib/neos (79)	36	9962	**58**	14578
miplib/other (405)	161	32009	**217**	50306
unibo (36)	8	**4764**	8	6826
market-split (20)	0	0	**8**	6075
opb/graphpart (31)	24	3795	**28**	715
opb/autocorr_bern (43)	8	**1838**	8	2180
opb/sporttournament (22)	11	3056	**13**	3089
opb/edgecross (19)	12	3433	**15**	6316
opb/pb (8)	0	0	0	0
opb/faclay (10)	1	**879**	1	1004
opb/other (6)	1	2	**3**	**4106**
primes/aim (48)	44	236	**46**	235
primes/jnh (16)	16	52	16	**42**
primes/ii (41)	34	5148	34	**5060**
primes/par (30)	20	**369**	20	426
primes/other (13)	5	1512	5	**976**
routing (15)	15	32	15	**28**
radar (12)	11	62	**12**	39
synthesis-ptl-cmos (10)	10	**17**	10	18
testset (6)	6	**8**	6	12
ttp (8)	2	12	2	**4**
vtxcov (15)	0	0	0	0
wnq (15)	0	0	0	0

6.3.4 Division of work between two components of PBO-IHS

Finally, we present our findings on how the processing work is divided between the MCHS solver and the PB oracle. Figure 6.6 (left) details the fraction of solving time spent on calls to the MCHS solver during the execution of PBO-IHS on the 898 of the instances solved within the time limit. Note that since calls to the MCHS solver and the PB oracle dominate the running time of PBO-IHS, the rest of the runtime is effectively spent on PB oracle calls. It can be observed that on most of the instances, over 80% of the overall solving time is spent on extracting cores: on 462 of the 898 instances, only 20% of the time was spent on calls to the MCHS solver, and on over 1/3 of the instances 99% of the overall solving time was spent on calls to the PB oracle. On the other hand, the cumulative runtime of the MCHS solver dominates on approximately 1/5 of the instances. From this we can conclude that in the current implementation of PBO-IHS, in most cases, the time it takes to extract cores is a more significant contributor to the execution time of PBO-IHS compared to the time it takes to find hitting sets over extracted cores.

If all constraints of a given PBO instance are seeded to the MCHS solver (recall Section 5.5), PBO-IHS relies entirely on the MCHS solver to solve the PBO instance. Figure 6.6 (right) shows the fractions of constraints that can be seeded over all benchmark instances in the sampled benchmark set. At least one constraint is seeded for 71.4% of the instances; at least half of all constraints are seeded for 41.6% of the instances; and all of the constraints are seeded for 33.7% of the instances. Note that we also observed that there are instances on which the cumulative runtime of the MCHS solver dominates even though all constraints are not seeded.



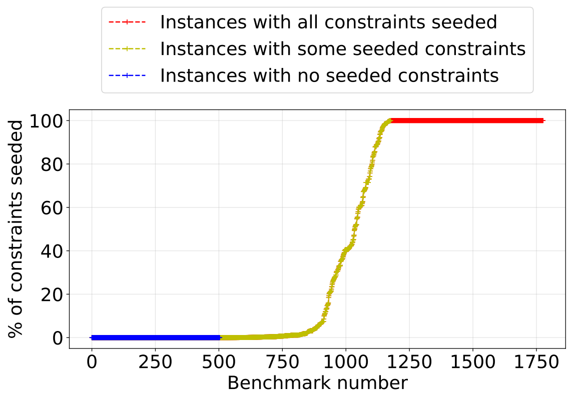


Figure 6.6: Left: Ratio of solving time spent by PBO-IHS on MCHS(O, K) calls for solved benchmarks. Right: Ratio of constraints seeded on all benchmarks.

7 Conclusion

In this thesis we described a novel approach to PBO solving through the use of implicit hitting sets and detailed an open-source implementation of the approach. The IHS framework allows for utilizing powerful optimization capabilities of a MIP solver while at the same time making use of recent developments in the realm of PB solving to perform fast core extraction. An instance of PBO is solved efficiently by using these two separate techniques in tandem. We presented a variety of techniques that enhance the IHS based PBO solving algorithm and evaluated their individual contribution to the performance of PBO-IHS. We compared the performance of PBO-IHS with the performance of other available PBO solvers on a large set of standard benchmarks. We found that in its best-performing configuration, PBO-IHS outperforms other published specialized PBO solvers. When compared to the state-of-the-art commercial MIP solver CPLEX, PBO-IHS was found to have complementary performance.

There are many directions for further research in order to potentially make the PBO-IHS solver perform better. In this thesis the cores extracted during the execution of PBO-IHS have unit coefficients and a unit right-hand bound when encoded as constraints. Using a PB solver that extracts core constraints with varying coefficients or a higher right-hand bound (like the one described in [33]) could lead to tighter lower bounds during the execution of PBO-IHS, possibly leading to faster solving times.

The reduced costs of the LP relaxation of the MCHS instance could be further utilized to obtain tighter bounds during the execution of PBO-IHS. This is motivated by other work where reduced costs have been succesfully utilized to solve NP-hard problems [96]. For example, reduced costs could be used as another heuristic for picking a subset from a set of cores retrieved via assumption shuffling as described in Section 5.2. Another example is to use reduced costs as weights for WCE instead of cost function coefficients as described in Section 5.4.

As PBO-IHS repeatedly calls two different black-box solvers during its execution, there is an opportunity to run these solvers in parallel. For example, a separate thread could call the MCHS solver while the disjoint phase is not finished with extracting cores. Assumption shuffling as described in Section 5.2 could also be executed in parallel. Other search improving techniques could be run as subroutines in parallel with the execution of the IHS-based PBO algorithm. One such subroutine could be a so-called solution-improving search, which is an optimization routine employed by some of the other PBO solvers [90, 67, 33]. The PBO-IHS solver provides both a lower and an upper bound for the cost of the optimal solution, which means that a binary search similar to the one described in [90] can be conducted in parallel. Another subroutine could probe the PB formula for unit cores by calling the PB solver with a unit assumption set to see if it returns `UNSAT`. Probing for unit cores could be done in parallel by brute-force, fixing variables of the PBO instance in the process.

Bibliography

[1] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger. A new look at BDDs for pseudo-Boolean constraints. *Journal of Artificial Intelligence Research*, 45:443–480, 2012. URL: https://doi.org/10.1613/jair.3653.

[2] M. Abseher, M. Gebser, N. Musliu, T. Schaub, and S. Woltran. Shift design with answer set programming. *Fundamenta Informaticae*, 147(1):1–25, 2016. URL: https://doi.org/10.3233/FI-2016-1396.

[3] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. URL: https://doi.org/10.1007/s12532-008-0001-1.

[4] L. Aksoy, E. A. C. da Costa, P. F. Flores, and J. Monteiro. Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1013–1026, 2008. URL: https://doi.org/10.1109/TCAD.2008.923242.

[5] S. Albert. *Solving Mixed Integer Linear Programs Using Branch and Cut Algorithm*. Master's thesis, North Carolina State University, 2006. URL: http://www4.ncsu.edu/~kksivara/masters-thesis/shon-thesis.pdf.

[6] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Generic ILP versus specialized 0-1 ILP: An update. In L. T. Pileggi and A. Kuehlmann, editors, *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*, pages 450–457. ACM / IEEE Computer Society, 2002. URL: https://doi.org/10.1145/774572.774638.

[7] F. A. Aloul, S. Z. H. Zahidi, A. Al-Farra, B. Al-Roh, and B. Al-Rawi. Solving the employee timetabling problem using advanced SAT & ILP techniques. *Journal of Computers*, 8(4):851–858, 2013. URL: http://www.jcomputers.us/index.php?m=content&c=index&a=show&catid=67&id=766.

[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, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. URL: https://doi.org/10.4230/LIPIcs.ICLP.2012.211.

[9] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In C. Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 399–404, 2009. URL: http://ijcai.org/Proceedings/09/Papers/074.pdf.

[10] F. Bacchus, A. Hyttinen, M. Järvisalo, and P. Saikko. Reduced cost fixing in MaxSAT. In C. Beck, editor, *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-66158-2_41.

[11] F. Bacchus, M. Järvisalo, and R. Martins. *Maximum satisfiability.* In *Handbook of Satisfiability.* Volume 336. Frontiers in Artificial Intelligence and Applications. IOS Press BV, 2021. Chapter 24, pages 929–991. URL: https://doi.org/10.3233/FAIA201008.

[12] M. Banbara, T. Soh, N. Tamura, K. Inoue, and T. Schaub. Answer set programming as a modeling language for course timetabling. *Theory and Practice of Logic Programming*, 13(4-5):783–798, 2013. URL: https://doi.org/10.1017/S1471068413000495.

[13] J. Berg and M. Järvisalo. Weight-aware core extraction in SAT-based MaxSAT solving. In J. C. Beck, editor, *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 652–670. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-66158-2_42.

[14] T. Berthold, G. Gamrath, A. Gleixner, S. Heinz, T. Koch, and Y. Shinano. Solving mixed integer linear and nonlinear problems using the SCIP Optimization Suite. Technical report 12-27, ZIB, Takustraße 7, 14195 Berlin, 2012.

[15] D. Bertsimas and S. S. Vempala. Solving convex programs by random walks. *Journal of the ACM*, 51(4):540–556, 2004. URL: https://doi.org/10.1145/1008731.1008733.

[16] P. Bieber, R. Delmas, and C. Seguin. DALculus - theory and tool for development assurance level allocation. In F. Flammini, S. Bologna, and V. Vittorini, editors, *Computer Safety, Reliability, and Security - 30th International Conference, SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings*, volume 6894 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-24270-0_4.

[17] G. Brewka, M. Diller, G. Heissenberger, T. Linsbichler, and S. Woltran. Solving advanced argumentation problems with answer set programming. *Theory and Practice of Logic Programming*, 20(3):391–431, 2020. URL: https://doi.org/10.1017/S1471068419000474.

[18] C. Buchheim, A. Wiegele, and L. Zheng. Exact algorithms for the quadratic linear ordering problem. *INFORMS Journal on Computing*, 22(1):168–177, 2010. URL: https://doi.org/10.1287/ijoc.1090.0318.

[19] M. R. Bussieck, A. S. Drud, and A. Meeraus. MINLPLib - A collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003. URL: https://doi.org/10.1287/ijoc.15.1.114.15159.

[20] D. Chai and A. Kuehlmann. A fast pseudo-Boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):305–317, 2005. URL: https://doi.org/10.1109/TCAD.2004.842808.

[21] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. A modular approach to MaxSAT modulo theories. In M. Järvisalo and A. V. Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 150–165. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-39071-5_12.

[22] 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, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. URL: https://doi.org/10.1145/800157.805047.

[23] W. J. Cook, C. R. Coullard, and G. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987. URL: https://doi.org/10.1016/0166-218X(87)90039-4.

[24] G. Cornuéjols and M. Dawande. A class of hard small 0-1 programs. *INFORMS Journal on Computing*, 11(2):205–210, 1999. URL: https://doi.org/10.1287/ijoc.11.2.205.

[25] I. I. Cplex. V12. 1: user's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[26] H. P. Crowder, E. L. Johnson, and M. W. Padberg. Solving large-scale zero-one linear programming problems. *Operational Research*, 31(5):803–834, 1983. URL: https://doi.org/10.1287/opre.31.5.803.

[27] A. S. S. da Graça. *Satisfiability-based Algorithms for Haplotype inference*. PhD thesis, Instituto Superior Técnico, 2011.

[28] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Operational Research*, 2(4):393–410, 1954. URL: https://doi.org/10.1287/opre.2.4.393.

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

[30] J. Davies and F. Bacchus. Postponing optimization to speed up MAXSAT solving. In C. Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-40627-0_21.

[31] J. Davies and F. Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In J. H. Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-23786-7_19.

[32] J. Devriendt, A. Gleixner, and J. Nordström. Learn to relax: integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 2021. ISSN: 1383-7133. URL: https://doi.org/10.1007/s10601-020-09318-x.

[33] J. Devriendt, S. Gocht, E. Demirovic, J. Nordström, and P. J. Stuckey. Cutting to the core of pseudo-Boolean optimization: combining core-guided search with cutting planes reasoning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3750–3758. AAAI Press, 2021. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16492.

[34] Dimacs. https://www.cs.princeton.edu/courses/archive/fall03/cs302/assignments/satisfaction/dimacs.pdf. Accessed: 29 April 2021.

[35] J. Dunagan and S. S. Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Mathematical Programming*, 114(1):101–114, 2008. URL: https://doi.org/10.1007/s10107-007-0095-7.

[36] K. Easton, G. L. Nemhauser, and M. A. Trick. The traveling tournament problem description and benchmarks. In T. Walsh, editor, *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*, volume 2239 of *Lecture Notes in Computer Science*, pages 580–584. Springer, 2001. URL: https://doi.org/10.1007/3-540-45578-7_43.

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

[38] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003. URL: https://doi.org/10.1016/S1571-0661(05)82542-3.

[39] M. Elf, M. Jünger, and G. Rinaldi. Minimizing breaks by maximizing cuts. *Operations Research Letters*, 31(3):343–349, 2003. URL: https://doi.org/10.1016/S0167-6377(03)00025-7.

[40] J. Elffers, J. Devriendt, S. Gocht, and J. Nordström. Roundingsat - the pseudo-Boolean solver powered by proof complexity! https://gitlab.com/MIAOresearch/roundingsat. Accessed 16 September 2021.

[41] J. Elffers and J. Nordström. Divide and conquer: towards faster pseudo-Boolean solving. In J. Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1291–1299. ijcai.org, 2018. URL: https://doi.org/10.24963/ijcai.2018/180.

[42] J. Farkas. Theorie der einfachen ungleichungen. *Journal für die reine und angewandte Mathematik*, 124:1–27, 1902. URL: http://eudml.org/doc/149129.

[43] 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, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2018. URL: https://doi.org/10.1007/978-3-319-94205-6_10.

[44] J. Franco and J. Martin. A history of satisfiability. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*. Volume 185, Frontiers in Artificial Intelligence and Applications, chapter 1, pages 3–55. IOS Press, 2009. URL: https://doi.org/10.3233/978-1-58603-929-5-3.

[45] P. Galinier, B. Jaumard, R. Morales, G. Pesan, E. Montréal, and C. Montreal. A constraint-based approach to the Golomb ruler problem, 2007.

[46] D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.

[47] M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele, and P. Veber. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In F. Lin, U. Sattler, and M. Truszczynski, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010.* AAAI Press, 2010. URL: http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1334.

[48] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991. URL: https://doi.org/10.1007/BF03037169.

[49] B. Ghaddar, M. F. Anjos, and F. Liers. A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. *Annals of Operations Research*, 188(1):155–174, 2011. URL: https://doi.org/10.1007/s10479-008-0481-4.

[50] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, and Y. Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. URL: https://doi.org/10.1007/s12532-020-00194-3.

[51] J. Gottlieb and L. Paulmann. Genetic algorithms for the fixed charge transportation problem. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pages 330–335. IEEE, 1998. URL: https://doi.org/10.1109/ICEC.1998.699754.

[52] A. Graça, J. Marques-Silva, I. Lynce, and A. L. Oliveira. Haplotype inference with pseudo-Boolean optimization. *Annals of Operations Research*, 184(1):137–162, 2011. URL: https://doi.org/10.1007/s10479-009-0675-4.

[53] M. Gwynne and O. Kullmann. *Attacking AES via SAT*. PhD thesis, 2010.

[54] S. S. Heragu and A. Kusiak. Machine layout problem in flexible manufacturing systems. *Operations Research*, 36(2):258–268, 1988. URL: https://doi.org/10.1287/opre.36.2.258.

[55] A. Hyttinen, P. Saikko, and M. Järvisalo. A core-guided approach to learning optimal causal graphs. In C. Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 645–651. ijcai.org, 2017. URL: https://doi.org/10.24963/ijcai.2017/90.

[56] A. Ignatiev, A. Previti, M. H. Liffiton, and J. Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In G. Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 173–182. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-23219-5_13.

[57] M. Janota, A. Morgado, J. F. Santos, and V. M. Manquinho. The Seesaw algorithm: function optimization using implicit hitting sets. In L. D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.CP.2021.31.

[58] S. Joshi, R. Martins, and V. M. Manquinho. Generalized totalizer encoding for pseudo-Boolean constraints. In G. Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-23219-5_15.

[59] A. P. Kamath, N. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992. URL: https://doi.org/10.1007/BF01581082.

[60] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984. URL: https://doi.org/10.1007/BF02579150.

[61] 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, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. URL: https://doi.org/10.1007/978-1-4684-2001-2_9.

[62] L. G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980. URL: https://doi.org/10.1016/0041-5553(80)90061-0.

[63] V. Klee and G. J. Minty. How good is the simplex algorithm. *Inequalities*, 3(3):159–175, 1972.

[64] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. D. Mittelmann, T. K. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011. URL: https://doi.org/10.1007/s12532-011-0025-9.

[65] R. Kolisch and A. Sprecher. PSPLIB - a project scheduling problem library: OR software - ORSEP operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997. ISSN: 0377-2217. URL: https://doi.org/10.1016/S0377-2217(96)00170-1.

[66] D. Le Berre, P. Marquis, and S. Roussel. Planning personalised museum visits. *Proceedings of the International Conference on Automated Planning and Scheduling*, 23(1):380–388, 2013. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/13587.

[67] D. Le Berre and A. Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010. URL: https://doi.org/10.3233/sat190075.

[68] F. Liers, E. Marinari, U. Pagacz, F. Ricci-Tersenghi, and V. Schmitz. A non-disordered glassy model with a tunable interaction range. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(05):L05003, 2010. ISSN: 1742-5468. URL: http://dx.doi.org/10.1088/1742-5468/2010/05/L05003.

[69] S. Malik, E. M. Sentovich, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Retiming and resynthesis: optimizing sequential networks with combinational techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):74–84, 1991. URL: https://doi.org/10.1109/43.62793.

[70] V. M. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for weighted Boolean optimization. In O. Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009. URL: https://doi.org/10.1007/978-3-642-02777-2_45.

[71] J. Marques-Silva and I. Lynce. On improving MUS extraction algorithms. In K. A. Sakallah and L. Simon, editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-21581-0_14.

[72] J. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999. URL: https://doi.org/10.1109/12.769433.

[73] R. Martins, V. M. Manquinho, and I. Lynce. Open-WBO: A modular MaxSAT solver, in C. Sinz and U. Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014. URL: https://doi.org/10.1007/978-3-319-09284-3_33.

[74] G. Mavrotas. Effective implementation of the epsilon-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2):455–465, 2009. URL: https://doi.org/10.1016/j.amc.2009.03.037.

[75] C. Michel and M. Rueher. Handling software upgradeability problems with MILP solvers. In I. Lynce and R. Treinen, editors, *Proceedings First International Workshop on Logics for Component Configuration, LoCoCo 2010, Edinburgh, UK, 10th July 2010*, volume 29 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–10, 2010. URL: https://doi.org/10.4204/EPTCS.29.1.

[76] H. D. Mittelmann. Testcases. http://plato.asu.edu/sub/testcases.html. Accessed: 29 April 2021.

[77] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. Wiley, 1988. URL: https://doi.org/10.1002/9781118627372.

[78] Y. Oren, M. Kirschbaum, T. Popp, and A. Wool. Algebraic side-channel analysis in the presence of errors. In S. Mangard and F. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2010. URL: https://doi.org/10.1007/978-3-642-15031-9_29.

[79] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[80] C. Pizzuti. Computing prime implicants by integer programming. In *Eigth International Conference on Tools with Artificial Intelligence, ICTAI '96, Toulouse, France, November 16-19, 1996*, pages 332–336. IEEE Computer Society, 1996. URL: https://doi.org/10.1109/TAI.1996.560473.

[81] B. C. Ribas, R. M. Suguimoto, R. A. N. R. Montaño, F. Silva, and M. A. Castilho. PBFVMC: A new pseudo-Boolean formulation to virtual-machine consolidation. In *Brazilian Conference on Intelligent Systems, BRACIS 2013, Fortaleza, CE, Brazil, 19-24 October, 2013*, pages 201–206. IEEE Computer Society, 2013. URL: https://doi.org/10.1109/BRACIS.2013.41.

[82] O. Roussel. Pseudo-Boolean competition 2016. http://www.cril.univ-artois.fr/PB16/. Accessed: 25 April 2021.

[83] O. Roussel and V. Manquinho. Input/output format and solver requirements for the competitions of pseudo-Boolean solvers, 2012. URL: http://www.cril.univ-artois.fr/PB12/format.pdf. Accessed: 25 April 2021.

[84] O. Roussel and V. Manquinho. *Pseudo-Boolean and cardinality constraints.* In *Handbook of Satisfiability, 2nd edition.* A. Biere, M. Heule, H. Van Maaren, and T. Walsh, editors. Frontiers in Artificial Intelligence and Applications. IOS Press BV, 2021. Chapter 28, pages 1087–1129. URL: https://doi.org/10.3233/FAIA201012.

[85] P. Saikko. *Implicit Hitting Set Algorithms for Constraint Optimization.* PhD thesis, University of Helsinki, 2019.

[86] P. Saikko. *Re-implementing and Extending a Hybrid SAT-IP Approach to Maximum Satisfiability.* Master's thesis, University of Helsinki, 2015. URL: https://helda.helsinki.fi/bitstream/handle/10138/159186/msc_thesis_saikko.pdf.

[87] P. Saikko, J. Berg, and M. Järvisalo. LMHS: A SAT-IP hybrid maxsat solver. In N. Creignou and D. Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016. URL: https://doi.org/10.1007/978-3-319-40970-2_34.

[88] P. Saikko, C. Dodaro, M. Alviano, and M. Järvisalo. A hybrid approach to optimization in answer set programming. In M. Thielscher, F. Toni, and F. Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018*, pages 32–41. AAAI Press, 2018. URL: https://aaai.org/ocs/index.php/KR/KR18/paper/view/18021.

[89] P. Saikko, J. P. Wallner, and M. Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In C. Baral, J. P. Delgrande, and F. Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth*

International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016, pages 104–113. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12812.

[90] M. Sakai and H. Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE Transactions on Information and Systems*, 98-D(6):1121–1127, 2015. URL: https://doi.org/10.1587/transinf.2014FOP0007.

[91] R. Sebastiani and P. Trentin. OptiMathSAT: A tool for optimization modulo theories. *Journal of Automated Reasoning*, 64(3):423–460, 2020. URL: https://doi.org/10.1007/s10817-018-09508-6.

[92] R. Shamir. The efficiency of the simplex method: a survey. *Management science*, 33(3):301–334, 1987.

[93] H. M. Sheini and K. A. Sakallah. Pueblo: A hybrid pseudo-Boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):165–189, 2006. URL: https://doi.org/10.3233/sat190020.

[94] P. Smirnov, J. Berg, and M. Järvisalo. Pseudo-Boolean Optimization by Implicit Hitting Sets. In L. D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, volume 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 51:1–51:20, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. URL: https://drops.dagstuhl.de/opus/volltexte/2021/15342.

[95] D. A. Spielman and S. Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004. URL: https://doi.org/10.1145/990308.990310.

[96] F. Trösser, S. de Givry, and G. Katsirelos. Improved acyclicity reasoning for Bayesian network structure learning with constraint programming. In Z. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4250–4257. ijcai.org, 2021. URL: https://doi.org/10.24963/ijcai.2021/584.

[97] J. P. Walser. 0-1 integer optimization benchmarks. https://www.ps.uni-saarland.de/~walser/benchmarks/benchmarks.html. Accessed: 29 April 2021.

[98] J. P. Walser. Solving linear pseudo-Boolean constraint problems with local search. In B. Kuipers and B. L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA*, pages 269–274. AAAI Press / The MIT Press, 1997. URL: http://www.aaai.org/Library/AAAI/1997/aaai97-042.php.

[99] K. Xu. Pseudo-Boolean (0-1 integer programming) benchmarks with hidden optimum solutions. http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/pb-benchmarks.htm. Accessed: 29 April 2021.

[100] C. Yang and M. J. Ciesielski. Synthesis for mixed CMOS/PTl logic. In I. Bolsens, editor, *2000 Design, Automation and Test in Europe (DATE 2000), 27-30 March 2000, Paris, France*, page 750. IEEE, 2000. URL: https://doi.org/10.1109/DATE.2000.840883.

[101] S. Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC), 1991.

[102] S. Zahidi, F. A. Aloul, A. Sagahyroon, and W. El-Hajj. Using SAT & ILP techniques to solve enhanced ILP formulations of the clustering problem in MANETS. In *8th International Wireless Communications and Mobile Computing Conference, IWCMC 2012, Limassol, Cyprus, August 27-31, 2012*, pages 1085–1090. IEEE, 2012. URL: https://doi.org/10.1109/IWCMC.2012.6314357.

[103] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in Boolean satisfiability solver. In R. Ernst, editor, *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2001, San Jose, CA, USA, November 4-8, 2001*, pages 279–285. IEEE Computer Society, 2001. URL: https://doi.org/10.1109/ICCAD.2001.968634.

Appendix A Problem domains

Table A.1 lists all problem domains from which benchmarks were used for the experiments in Section 6. Each problem domain entry includes the name of the problem domain with the number of benchmarks per problem domain stated in parentheses. The entry is followed by a brief description on the nature of the problem and relevant citations from which instances of the problem domain have originated from. The table has (?) marked for the description and citation entry in case we could not verify the source of the given problem domain.

Table A.1: Problem domains from the benchmark set used for experiments. Problem domain name is followed by the number of benchmarks that are present in the domain (#). Description and citation entries are marked with (?) if we could not verify the source of the given problem domain.

Problem domain (#)	**Description**	**Citation**
10orplus/9orless (156)	Upgradeability problem – finding the best solution according to some criteria to install, remove or upgrade packages in a given installation	[75]
caixa (24)		
rand.*list (118)		
BA (1440)	Planning personalized Museum visits given a set of preferences and constraints – Palais des Beaux Arts de Lille	[66]
NG (960)	Planning personalized Museum visits given a set of preferences and constraints – National Gallery of London	[66]
BioRepair (30)	Repairing large-scale biological networks (encoded to PBO from Answer Set Programming)	[47]
Metro (30)	Problems related to transport systems (encoded to PBO from Answer Set Programming)	[17]
ShiftDesign (30)	Minimizing the number of workshifts while reducing over- and understaffing (encoded to PBO from Answer Set Programming)	[2]
Timetabling (30)	Assigning a number of lectures to a limited set of timeslots and rooms (encoded to PBO from Answer Set Programming)	[12]
EmployeeScheduling (14)	Assigning employees into a given set of shifts over a fixed period of time while meeting the preferences and organizational work regulations of the employees	[7]
MANETs (150)	Clustering Problem in Mobile Ad-Hoc Networks to maximize the lifetime of the network	[102]
area_* (59)	Minimization of area in multiple constant multiplications for FIR digital filters	[4]
trarea_ac (18)		
aries-da_nrp (70)	?	?
bsg (60)	?	?
mis/mds (120)	?	?
course-ass (6)	Course allocation for the law school of the Saarland University	[97]
decomp (10)	?	?
data (68)	Problems of various problem domains retireved from Netlib repository	[46]
dt-problems (60)	?	?
domset (15)	?	?
factor (186)	Linearized number factorization problems	[83]
factor-mod-B (225)		
featureSubscription (20)	?	?
frbXX-XX-opb (40)	Generated satisfiable PBO instances	[99]

Table A.1: (continued) Problem domains from the benchmark set used for experiments. Problem domain name is followed by the number of benchmarks that are present in the domain (#). Description and citation entries are marked with (?) if we could not verify the source of the given problem domain.

Problem domain (#)	**Description**	**Citation**
fctp (4)	Fixed charge transportation problem – minimizing the shipping cost between plants and customers depending on the transported amount of a single commodity	[51]
flexray (9)	?	?
fome (3)	?	?
graca (100)	Haplotype inference for identifying genetic variations	[27]
haplotype (8)		[52]
golomb-rulers (34)	Finding smallest Golomb rulers	[45]
garden (7)	?	?
hw32/hw64/hw128 (27)	Placing a set of virtual machines in a set of hardware in such a way that the workload on hardware is increased and the virtual machines operate more energy-efficiently	[81]
jXXopt (2040)	Generated job-scheduling problems from the PSPLib library	[65]
keeloq_tasca (4)	Tolerant Algebraic Side-Channel Analysis – attacks on the Keeloq cipher	[78]
kullmann (7)	Attacking AES block cipher	[53]
lion9-single-obj (1513)	Optimizing the costs for quality assurance of hardware and software functions of an aircraft	[16]
logic-synthesis (74)	Implementing a logical specification into a circuit	[69, 101]
miplib/neos (81)	Unknown problems from the MIPLIB library that were gathered from the NEOS server	[64]
miplib/other (457)	Problems of various problem domains retireved from the MIPLIB library	[64]
market-split (20)	Generated PBO instances for market-split problem	[24]
opb/graphpart (31)	Graph partitioning problem	[49]
opb/autocorr_bern (43)	Degree-four model from theoretical physics for low autocorrelated binary sequences	[68]
opb/sporttournament (22)	Minimizing breaks in sports tournaments via max-cut encoding	[39]
opb/edgecross (19)	Edge-crossing minimization in bipartite graphs	[18]
opb/pb (8)	General quadratic assignment problem that was linearized	[74]
opb/faclay (10)	Arranging single row facilities to minimize the total weighted sum of the center-to-center distances between all pairs of facilities	[54]
opb/other (6)	Miscellaneous benchmarks from the MINLPLib library	[19]
primes/aim (48)	Computing prime implicants from a set of artificially generated 3-sat instances with exactly one satisfying assignment	[80, 34]
primes/jnh (16)	Computing prime implicants from a set of random instances generated by John Hooker	[80, 34]
primes/ii (41)	Computing prime implicants from inductive inference instances	[80, 34, 59]
primes/par (30)	Computing prime implicants from instances that arise from a problem in learning the parity function	[80, 34]
primes/other (13)	Computing prime implicants from other miscellaneous CNF instances	[80, 34]
routing (15)	Global routing of a random set of n two-pin connections over a two-dimensional grid of cells, where the capacity of each intercell channel is limited	[6]
radar (12)	Allocate a number of radar stations for observation of a geographic area, such that each point is observed by at least three stations	[98]
synthesis-ptl-cmos (10)	Performing logic synthesis for PTL/CMOS circuits	[100]
testset (6)	?	?
ttp (8)	Sports timetabling problems – scheduling sports matches in consideration with the travel time and the playing schedule of the teams	[36]
unibo (36)	Problems of various problem domains supplied by the University of Bologna	[76]
vtxcov (15)	?	?
wnq (15)	?	?