Pseudo-Boolean Optimization by Implicit Hitting Sets

Pavel Smirnov
HIIT, Department of Computer Science, University of Helsinki, Finland

Jeremias Berg
HIIT, Department of Computer Science, University of Helsinki, Finland

Matti Järvisalo
HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract

Recent developments in applying and extending Boolean satisfiability (SAT) based techniques have resulted in new types of approaches to pseudo-Boolean optimization (PBO), complementary to the more classical integer programming techniques. In this paper, we develop the first approach to pseudo-Boolean optimization based on instantiating the so-called implicit hitting set (IHS) approach, motivated by the success of IHS implementations for maximum satisfiability (MaxSAT). In particular, we harness recent advances in native reasoning techniques for pseudo-Boolean constraints, which enable efficiently identifying inconsistent assignments over subsets of objective function variables (i.e. unsatisfiable cores in the context of PBO), as a basis for developing a native IHS approach to PBO, and study the impact of various search techniques applicable in the context of IHS for PBO. Through an extensive empirical evaluation, we show that the IHS approach to PBO can outperform other currently available PBO solvers, and also provide a complementary approach to PBO when compared to classical integer programming techniques.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Constraint and logic programming

Keywords and phrases constraint optimization, pseudo-Boolean optimization, implicit hitting sets

Digital Object Identifier 10.4230/LIPIcs.CP.2021.36

Supplementary Material Source code and experiment data are available at `https://bitbucket.org/coreo-group/pbo-ihs-solver/`

Funding Work financially supported by Academy of Finland under grants 322869 and 342145.

Acknowledgements The authors thank Paul Saikko for his initial implementation work on PBO-IHS.

1 Introduction

Declarative approaches are central in efficiently solving various types of NP-hard real-world optimization problems. Indeed various constraint optimization paradigms have been developed, ranging from mixed integer linear programming (MIP) [32] to finite-domain constraint optimization [34] and Boolean satisfiability (SAT) based maximum satisfiability (MaxSAT) [3] and its extensions to e.g. optimization modulo theories and MaxSMT [11, 41]. Each of the paradigms offer distinct features in terms of the declarative language used and the underlying algorithmic approach, ranging from branch-and-cut in MIP to the unsatisfiability-based search through iterative applications of SAT solvers in MaxSAT.

Pseudo-Boolean (PB) constraints [36] constitute an interesting constraint language for modelling and solving optimization problems. Also known as 0-1 linear constraints, stated as linear inequalities with integer coefficients over binary variables, pseudo-Boolean constraints constitute a central fragment of integer programming. However, PB constraints can also be viewed as natural generalizations of conjunctive normal form clausal constraints [5, 36].

Taking this view, effective specialized decision procedures have been developed for PB by lifting search techniques from the realm of SAT solving, boosted with additional inference techniques which lift the theoretical efficiency of PB solvers beyond that of standard SAT solvers [24, 10, 42, 12]. For a recent overview of such conflict-driven pseudo-Boolean solving, we refer the reader to [9]. Recent work on extending these techniques from decision to optimimization problems by harnessing search techniques from both core-guided MaxSAT solving [21] and linear programming [20] have been shown to hold promise as an alternative approach to pseudo-Boolean optimization (PBO) complementing the more classical MIP solving techniques [33].

Building on these recent developments, in this work we develop an alternative approach to PBO drawing from both advances in PB solving and IP solving. In particular, motivated by the success of the so-called implicit hitting set (IHS) approach to MaxSAT [16, 17, 18, 37] as a current state-of-the-art MaxSAT solving approach alongside the core-guided approach, we develop a first instantiation of an IHS PBO solver. While the general IHS solving framework has been shown to be applicable in a range of settings [18, 19, 28, 39, 27, 25, 38], we are not aware of earlier work studying the applicability of IHS in the context of PBO. For realizing a competitive IHS PBO solver, we harness recent advances in native reasoning techniques for pseudo-Boolean constraints, which enable efficiently identifying inconsistent assignments over subsets of objective function variables [20], i.e., unsatisfiable cores in the context of PB. As the other major component, we employ integer programming and linear programming for hitting set computations over iteratively accumulated unsatisfiable cores as well as for integrating bounds-based inference techniques [14, 2]. We provide results from an extensive empirical evaluation of our implementation of the IHS approach to PBO, 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 of the empirical performance of the IHS PBO solver. It turns out that, overall, our IHS PBO solver outperforms earlier advances in specialized PBO solving, and shows complementary performance depending on the problem domains considered with respect to both other specialized PBO solvers and a commercial MIP solver.

2 Preliminaries

A binary variable x has the domain $\{0, 1\}$. A literal l over a variable x is either x or $\overline{x} \equiv (1-x)$. A pseudo-Boolean (PB) constraint C is a 0-1 integer linear inequality $\sum_i a_i l_i \geq B$ over literals l_i. The set of variables appearing in C is $\textsc{var}(C)$. We assume w.l.o.g. that all PB constraints are in normalized form, i.e., that each variable appearing in it is distinct and that the coefficients a_i and bound B are non-negative integers. We use $l = 0$ as shorthand for the constraints $l \geq 0$ and $-l \geq 0$ (rewritten in normal form). An assignment $\tau\colon \textsc{var}(C) \to \{0, 1\}$ is extended to literals by $\tau(\bar{l}) = 1 - \tau(l)$. An assignment τ satisfies C ($\tau(C) = 1$) if $\sum_i a_i \tau(l_i) \geq B$. When convenient we treat an assignment τ over a set X of variables as a set of literals $\tau = \{x \mid x \in X \wedge \tau(x) = 1\} \cup \{\overline{x} \mid x \in X \wedge \tau(x) = 0\}$.

A PB formula $F = \{C_1, \ldots, C_n\}$ is a set of PB constraints. We denote by $\textsc{var}(F)$ the set of variables appearing in the constraints of F. An assignment $\tau\colon \textsc{var}(F) \to \{0, 1\}$ is a solution to F if it satisfies all constraints in F. We use $\tau(F) = 1$ to denote that τ is a solution to F; $\tau(F) = 0$ denotes that τ is not a solution to F.

An instance $\mathcal{F}$ of the pseudo-Boolean optimization problem (PBO) consists of a PB formula $\textsc{constraints}(\mathcal{F})$ and an objective function $\mathrm{O}^{\mathcal{F}} \equiv \sum_i w_i l_i$ where each l_i is a literal over a variable $x_i \in \textsc{var}(\textsc{constraints}(\mathcal{F}))$ and w_i its non-negative integer weight. When clear from context, we use $\mathcal{F}$ and $\textsc{constraints}(\mathcal{F})$ interchangeably and drop the superscript from $\mathrm{O}^{\mathcal{F}}$. We will sometimes abuse notation and treat O as either a set of literals or a set of weight-literal tuples, i.e., write $l \in \mathrm{O}$ and $(w, l) \in \mathrm{O}$ to obtain either literals or weight-literal pairs from O. The set of variables appearing in O is $\textsc{var}(\mathrm{O})$. The value of O under an assignment $\tau \colon \textsc{var}(\mathrm{O}) \to \{0,1\}$ is $\mathrm{O}(\tau) = \sum_i w_i \tau(l_i)$. A solution τ to $\mathcal{F}$ is optimal if it minimizes $\mathrm{O}(\tau)$ over all solutions to $\mathcal{F}$. The PBO problem consists of finding an optimal solution to a given PBO instance.

The approach to computing optimal solutions of PBO instances presented in this work makes use of so called *core constraints* and *hitting sets.*

▶ **Definition 1.** *A constraint $C = \sum_i a_i l_i \geq B$ is a* core constraint *of $\mathcal{F}$ if: i) $\textsc{var}(C) \subset \textsc{var}(O^{\mathcal{F}})$ and ii) $(\tau(\mathcal{F}) = 1) \to (\tau(C) = 1)$ holds for all solutions to $\mathcal{F}$.*

In words, a core constraint of an instance $\mathcal{F}$ is a constraint over the variables in the objective function that is satisfied by any solution to $\mathcal{F}$.

▶ **Example 2.** Let $0 < r < n$ be two integers and consider the instance $\mathcal{F}^{n,r}$ with the constraints $\{\sum_{i=1}^{n} b_i \geq r\}$ and objective function $\mathrm{O} \equiv \sum_{i=1}^{n} b_i$. Now $\textsc{var}(\mathcal{F}) = \{b_1, \ldots, b_n\}$ and any assignment τ that assigns at least r variables in $\textsc{var}(\mathcal{F})$ to 1 is a solution to $\mathcal{F}$. The assignment τ^o that sets $\tau^o(b_j) = 1$ for $j = 1 \ldots r$ and $\tau^o(b_k) = 0$ for $k = r+1 \ldots n$ is an optimal solution to $\mathcal{F}^{n,r}$. The cost of τ^o (and thus the cost of $\mathcal{F}^{n,r}$) is $\mathrm{O}(\tau^o) = \mathrm{O}(\mathcal{F}^{n,r}) = r$. The constraint $\sum_{i=1}^{n} b_i \geq t$ is a core constraint of $\mathcal{F}$ for all $t = 1 \ldots r$, as is $C = \sum_{b \in S} b \geq 1$ for any set $S \subset \mathrm{O}$ of literals containing at least $n - r + 1$ variables. To see why C is a core constraint, notice that any solution τ to $\mathcal{F}$ sets at least r of the n literals in O to 1 will also set at least one literal in S to 1 as well.

Given a set $\mathcal{C}$ of core constraints of an instance $\mathcal{F}$, we say that an assignment $\gamma \colon \textsc{var}(\mathrm{O}) \to \{0,1\}$ that satisfies $\mathcal{C}$ is a *hitting set* of $\mathcal{C}$. A hitting set γ^o is optimal if $\mathrm{O}(\gamma^o) \leq \mathrm{O}(\gamma)$ holds for all hitting sets of $\mathcal{C}$. The term hitting set stems from an important special case of core constraints, namely, those of form $C = \sum l \geq 1$. Such constraints are satisfied by setting at least one $l \in C$ to 1, thus *hitting* that constraint. For our purposes, a central property of hitting sets is that they provide lower bounds on $\mathrm{O}(\mathcal{F})$.

▶ **Proposition 3.** *Let γ^o, $\mathcal{C}$ and $\mathcal{F}$ be as above. Then $O(\gamma^o) \leq O(\mathcal{F})$.*

Proof. Let τ be an optimal solution of $\mathcal{F}$. Then $\tau(\mathcal{C}) = 1$ by the definition of a core constraint and $\mathrm{O}(\gamma^o) \leq \mathrm{O}(\tau) = \mathrm{O}(\mathcal{F})$ by the optimality of γ^o. ◀

3 Implicit Hitting Sets for Pseudo Boolean Optimization

Algorithm 1 details the `PBO-IHS` algorithm for computing an optimal solution to a PBO instance $\mathcal{F}$. In short, the algorithm works by iteratively refining an upper and lower bound on $\mathrm{O}(\mathcal{F})$, represented in the pseudocode by UB and LB, respectively. The algorithm also maintains a witness for the upper bound in the form of an assignment τ_{best} for which $\mathrm{O}(\tau_{best}) = UB$. The search terminates when $LB = UB$ at which point τ_{best} is returned.

During initialization (Lines 2-5) the lower bound LB and set $\mathcal{C}$ of core constraints of $\mathcal{F}$ are initialized to 0 and $\emptyset$, respectively. Additionally, an upper bound UB (as well as its witness τ_{best}) is obtained by invoking a PB solver via the function `PB-Solve` on the constraints of $\mathcal{F}$. The call to `PB-Solve` returns a boolean sat? indicating whether or not the constraints in $\mathcal{F}$ are satisfiable and a solution of $\mathcal{F}$ if they are. Note that, if the constraints of $\mathcal{F}$ are not satisfiable, then there do not exist any solutions to $\mathcal{F}$, so `PBO-IHS` terminates. Afterwards the main search loop is started.

```
PBO-IHS($\mathcal{F}$)
    Input: A PBO instance $\mathcal{F}$
    Output: An optimal solution $\tau$
    $(\tau_{best}, sat?) \leftarrow$ PB-Solve($\mathcal{F}$)
    if not sat? then
        return "no feasible solutions"
    $UB \leftarrow \mathrm{O}(\tau_{best})$; $LB \leftarrow 0$; $\mathcal{C} \leftarrow \emptyset$
    while TRUE do
        $\gamma \leftarrow$ Min-Hs($\mathrm{O}, \mathcal{C}$)
        $LB \leftarrow \mathrm{O}(\gamma)$
        if $UB = LB$ then break ;
        $\mathcal{C} \leftarrow \mathcal{C} \cup$ Extract-Cores($\gamma, UB, \tau_{best}, \mathcal{F}$);
        if $UB = LB$ then break;
    return $\tau_{best}$
```

Algorithm 1 The base IHS algoritm for PBO

```
Min-Hs($\mathrm{O}, \mathcal{C}$):
minimize:    $\sum_{(w,l)\in \mathrm{O}} w \cdot l$
subject to:
    $C$                 $\forall C \in \mathcal{C}$
    $l \in \{0,1\}$     $\forall (w,l) \in \mathrm{O}$
return:
    $\{l \mid l$ set to 1 in opt. soln$\} \cup$
    $\{\bar{l} \mid l$ set to 0 in opt. soln$\}$
```

(a) An IP for computing an optimal hitting set over a set of core constraints

Figure 1 The implicit hitting set approach to PBO

```
Extract-Cores($\gamma, UB, \tau_{best}, \mathcal{F}$)
    $\mathcal{A} = \{l \mid l \in \mathrm{O} \wedge \gamma(l) = 0\}$;
    $\mathcal{C}_n \leftarrow \emptyset$;
    while TRUE do
        $(sat?, \kappa, \tau) \leftarrow$ PB-Solve-A$(\mathcal{F}, \mathcal{A})$;
        if $(sat?)$ then
            if $\mathrm{O}(\tau) < UB$ then $\tau_{best} \leftarrow \tau$; $UB \leftarrow \mathrm{O}(\tau)$;
            return $\mathcal{C}_n$;
        else $\mathcal{C}_n \leftarrow \mathcal{C}_n \cup \{\sum_{l\in\kappa} l \geq 1 \mid l \in \kappa\}$; $\mathcal{A} \leftarrow \mathcal{A} - \kappa$;
```

Algorithm 2 Extracting multiple core constraints from a single hitting set.

During each iteration of the loop (Lines 6-11), the lower bound is refined by computing an optimal hitting set γ over $\mathcal{C}$ on Line 8. In our implementation, the hitting set is computed by solving the integer program `Min-Hs` detailed in Figure 1a. If the new LB matches the known UB the algorithm terminates on Line 9. Otherwise, the upper bound UB and set $\mathcal{C}$ are next refined by the function `Extract-Cores` detailed in Algorithm 2. After refining the upper bound and extracting new core constraints, the termination criteria is again checked. If the new UB matches the current LB, the algorithm terminates. Otherwise, the loop reiterates.

`Extract-Cores` computes new core constraints of $\mathcal{F}$ by invoking a PB solver on the constraints of $\mathcal{F}$ under a set $\mathcal{A}$ of *assumptions*. The inputs to `Extract-Cores` is the current hitting set γ of $\mathcal{C}$, the upper bound UB, its witness τ_{best} and the constraints of $\mathcal{F}$. The function initialises a set $\mathcal{A}$ to contain all literals in O set to 0 by γ. In other words, initially the set $\mathcal{A}$ contains all literals of O that do not incur cost in γ. A set $\mathcal{C}_n$ of new core constraints is also initialized to $\emptyset$. New core constraints are then computed by invoking a PB solver via the function `PB-Solve-A`. The function takes as input a set $\mathcal{F}$ of constraints and a set $\mathcal{A}$ of assumptions and then solves the formula $\mathcal{F} \cup \{l = 0 \mid l \in \mathcal{A}\}$. There are two options, either the formula is satisfiable (*sat?* is true), or it is not (*sat?* is false). In the first case, the call to `PB-Solve-A` returns a solution τ to $\mathcal{F}$ that sets $\tau(l) = 0$ for all $l \in \mathcal{A}$. Then $\mathrm{O}(\tau)$ is compared to UB which is updated if needed. Afterwards `Extract-Cores` terminates and returns the set $\mathcal{C}_n$ of new core constraints found. In the second case, $\kappa \subset \mathcal{A}$ is a set of literals for which $\mathcal{F} \cup \{l = 0 \mid l \in \kappa\}$ is also unsatisfiable. This implies that $\sum_{l \in \kappa} l \geq 1$ is a core-constraint of $\mathcal{F}$ so it is added to $\mathcal{C}_n$. The literals in κ are then removed from $\mathcal{A}$ and the loop reiterated.

The following theorem establishes the correctness of `PBO-IHS`.

▶ **Theorem 4.** *Given an input PBO instance $\mathcal{F}$* `PBO-IHS` *terminates and returns an optimal solution τ to $\mathcal{F}$.*

Proof. (Sketch) To show that τ is optimal we note that $\mathrm{O}(\tau) = \mathrm{O}(\gamma)$ for an optimal hitting set γ over a set $\mathcal{C}$ of core constraints of $\mathcal{F}$, which by Proposition 3 implies $\mathrm{O}(\tau) \leq \mathrm{O}(\mathcal{F})$.

To show that `PBO-IHS` terminates, we first show that each call to `Extract-Cores` terminates. This follows from each invocation of `PB-Solve-A` either resulting in termination of `Extract-Cores`, or elements being removed from $\mathcal{A}$ and the fact that, by the check on Line 2, the call `PB-Solve-A`$(\mathcal{F}, \emptyset)$ returns satisfiable.

For the final part of the argument, we say that a hitting set γ returned on Line 7 is feasible if `PB-Solve-A`$(\mathcal{F}, \{l \mid l \in \mathrm{O} \wedge \gamma(l) = 0\})$ is satisfiable, otherwise it is infeasible. We note that, as soon as a feasible hitting set γ is computed by `Min-Hs`, `PB-Solve-A` will find a solution τ for which $\mathrm{O}(\tau) = \mathrm{O}(\gamma) = LB$ in the first iteration of `Extract-Cores` and `PBO-IHS` will terminate. As there only are a finite number of possible hitting sets, we thus only need to show that a fixed infeasible hitting set γ^I can be computed at most once by `Min-Hs`. This follows from the fact that γ^I being infeasible implies that the invocation of `Extract-Cores` will add (at least) one new core constraint $\sum_{l \in \kappa} l \geq 1$ for some $\kappa \subset \mathrm{O} \setminus \{l \mid \gamma(l) = 1\}$ into the set $\mathcal{C}$. Thus γ^I will not be a hitting set in subsequent iterations. ◀

We end this section with an example demonstrating the execution of `PBO-IHS`.

▶ **Example 5.** Invoke `PBO-IHS` on the instance $\mathcal{F}^{5,2}$ from Example 2 with $n = 5$ and $r = 2$. Assume that the first solution obtained on on line 2 is $\tau_{best} = \{b_1, b_2, b_3, b_4, \overline{b_5}\}$. The initial upper bound is set to $UB = \mathrm{O}(\tau_{best}) = 4$. In the first iteration of the search loop, there are no core constraints to satisfy. As such, the first call to `Min-Hs` returns $\gamma = \{\overline{b_1}, \overline{b_2}, \overline{b_3}, \overline{b_4}, \overline{b_5}\}$. As $\mathrm{O}(\gamma) = 0$, the lower bound LB is not improved and the algorithm moves on to invoke `Extract-Cores`. The set $\mathcal{A}$ is initialized to $\{b_1, b_2, b_3, b_4, b_5\}$. The first call to `PB-Solve-A` returns unsatisfiable. There are a number of subsets of the assumptions that could be returned, let $\kappa = \{b_1, b_2, b_3, b_4\}$ be the one obtained. Before the next call, the set $\mathcal{A}$ is refined to $\{b_5\}$ and the core constraint $\sum_{i=1}^{4} b_i \geq 1$ is added to $\mathcal{C}_n$. The next call returns satisfiable, returning for example the solution $\tau = \{b_1, b_2, b_3, \overline{b_4}, \overline{b_5}\}$. The solution has $\mathrm{O}(\tau) = 3$ so the upper bound and τ_{best} are updated before `Extract-Cores` terminates. At this point $UB = 3 \neq 0 = LB$ so `PBO-IHS` does not terminate.

In the next iteration, the call to `Min-Hs` is done with $\mathcal{C} = \{\sum_{i=1}^{4} b_i \geq 1\}$. Assume the call returns $\gamma = \{b_1, \overline{b_2}, \overline{b_3}, \overline{b_4}, \overline{b_5}\}$. The lower bound LB is now updated to 1 and the function `Extract-Cores` is again invoked. This time around, the first call to `PB-Solve-A` is done with $\mathcal{A} = \{b_2, b_3, b_4, b_5\}$. The first call is unsatisfiable, the only subset of assumptions that can be returned is $\kappa = \{b_2, b_3, b_4, b_5\}$. The next call to `PB-Solve-A` will return satisfiable. Assume that this time a solution $\tau = \{b_1, b_2, \overline{b_3}, \overline{b_4}, \overline{b_5}\}$ is returned. The solution has $\mathrm{O}(\tau) = 2$ so the upper bound is again updated.

At this point, `PBO-IHS` has found an optimal solution of $\mathcal{F}^{5,2}$. However, since $UB = 2 > 1 = LB$, the algorithm does not terminate. Informally speaking, the algorithm has found an optimal solution, but not proven its optimality. The "proof" of optimality is obtained once `Min-Hs` returns a hitting set γ with $\mathrm{O}(\gamma) = 2$, which in turn happens after enough core constraints have been extracted for $\mathcal{C} = \{(b_1 + b_2 + b_3 + b_4 \geq 1), (b_1 + b_2 + b_3 + b_5 \geq$

$1), (b_1 + b_2 + b_4 + b_5 \geq 1), (b_1 + b_3 + b_4 + b_5 \geq 1), (b_2 + b_3 + b_4 + b_5 \geq 1)\}$. In other words for each b_i $\mathcal{C}$ should contain at least one constraint in which b_i does not appear. Then `Min-Hs` returns a hitting set γ with $\mathrm{O}(\gamma) = 2$, which updates $LB = 2$ and allows the algorithm to terminate.

4 Search Techniques and Refinements

We move on to describing a number of refinements and additional heuristics to `PBO-IHS`. We will later on empirically evaluate the impact of each of the techniques on the runtime performance of `PBO-IHS`.

Many of the refinements we consider are based on techniques first proposed for the IHS algorithm in the context of MaxSAT. These are motivated by the fact that, in order for `PBO-IHS` to terminate, the lower bound LB needs to be set to the optimal cost $\mathrm{O}(\mathcal{F})$ of the instance $\mathcal{F}$ that is being solved. This in turns means that the `Min-Hs` subroutine should compute a hitting set γ for which $\mathrm{O}(\gamma) = \mathrm{O}(\mathcal{F})$. In fact, by adapting a well known result from MaxSAT, we can show that there are families of instances on which `PBO-IHS` as presented in Section 3 requires an exponential number of core constraints from `Extract-Cores` in order to terminate.

▶ **Proposition 6.** *(Adapted from [16]) For every even $n \in \mathbb{N}$ there exists a PBO instance $\mathcal{F}_n$ for which* `Extract-Cores` *needs to extract $\Omega(2^n)$ core-constraints before* `PBO-IHS` *terminates.*

Proof. (Sketch) Let $r = n/2$ and $\mathcal{F}_n = \mathcal{F}^{n,r}$ from Example 2 and, following similar reasoning as in [16] in the context of MaxSAT, to show that in order for `Min-Hs` to compute a hitting set γ with $\mathrm{O}(\gamma) = n/2 = \mathrm{O}(\mathcal{F}^{n,r})$, `Extract-Cores` needs to extract at least one core-constraint of form $\sum_{l \in S} l \geq 1$ for each subset $S \subset \mathrm{O}$ with $n - r + 1$ literals.

More precisely, if there exists a subset $S_p \subset \mathrm{O}$ with $n - r + 1$ elements for which $\left(\sum_{l \in S_p} l \geq 1\right) \notin \mathcal{C}$, then the solution $\gamma = \{\bar{l} \mid l \in S_p\} \cup \{l \mid l \notin S_p\}$ is a hitting set over $\mathcal{C}$ that has $\mathrm{O}(\gamma) = n - (n - r + 1) = r - 1 < r = \mathrm{O}(\mathcal{F}^{n,r})$. As a consequence, the minimum-cost hitting set γ computed by `Min-Hs` will have $\mathrm{O}(\gamma) < \mathrm{O}(\mathcal{F}^{n,r})$ and the algorithm will not terminate. In other words, `Extract-Cores` will need to extract at least $\binom{n}{r+1}$ core constraints before `Min-Hs` computes a hitting set γ with $\mathrm{O}(\gamma) = \mathrm{O}(\mathcal{F}^{n,r})$. ◀

In light of Proposition 6 we expect any technique for deriving more core constraints of an instance to improve on the empirical performance of `PBO-IHS`. In this work, we consider the following techniques.

4.1 Constraint Seeding

In constraint seeding, the input instance $\mathcal{F}$ is scanned for constraints that only contain variables that appear in the objective function. Such constraints trivially satisfy the second requirement of Definition 1 and as such are core constraints of $\mathcal{F}$. Any such constraints are added to $\mathcal{C}$ prior to starting the main search loop (Lines 6-11 of Algorithm 1). While a similar technique is employed in MaxSAT solving, in the context of PBO we can show that constraint seeding can have a significant effect on the number of core constraints that `PBO-IHS` needs to extract before termination.

▶ **Example 7.** Consider again the instance $\mathcal{F}^{5,2}$ from Example 2. On this instance constraint seeding is able to detect the core constraint $C = \sum_{i=1}^{5} b_i \geq 2$ and add it to $\mathcal{C}$. Assume that the first solution τ_{best} obtained on line 2 is $\{b_1, b_2, b_3, b_4, \overline{b_5}\}$ implying an initial UB of 4. In the

```
1  Extract-Cores-WCE($\gamma, UB, \tau_{best}, \mathcal{F}$)
2      $\mathcal{C}_n \leftarrow \emptyset$; $\mathcal{W} \leftarrow \emptyset$;
3      for $(w, l) \in O$ do
4          if $\gamma(l) = 1$ then $\mathcal{W}(l) = 0$;
5          else $\mathcal{W}(l) = w$;
6      while TRUE do
7          $(sat?, \kappa, \tau) \leftarrow$ PB-Solve-A$\big(\mathcal{F}, \{l \in \mathrm{O} \mid \mathcal{W}(l) > 0\}\big)$;
8          if $(sat?)$ then
9              if $\mathrm{O}(\tau) < UB$ then $\tau_{best} \leftarrow \tau$; $UB \leftarrow \mathrm{O}(\tau)$;
10             return $\mathcal{C}_n$;
11         else
12             $\mathcal{C}_n \leftarrow \mathcal{C}_n \cup \{\sum_{l \in \kappa} l \geq 1 \mid l \in \kappa\}$;
13             $w^\kappa = \min_{l \in \kappa}\{\mathcal{W}(l)\}$;
14             for $l \in \kappa$ do $\mathcal{W}(l) \leftarrow \mathcal{W}(l) - w^\kappa$;
```

Algorithm 3 Computing multiple core constraints with weight-aware core extraction.

first iteration of the search loop, the core-constraint C added by seeding results in the hitting set γ computed on Line 7 assigning at least two variables to 1. Assume $\gamma = \{b_1, b_2, \overline{b_3}, \overline{b_4}, \overline{b_5}\}$. The LB is then refined to 2 and the function `Extract-Cores` invoked. In the first iteration of `Extract-Cores`, the function `PB-Solve-A` is invoked with $\mathcal{A} = \{b_3, b_4, b_5\}$. The result is satisfiable and the function returns the assignment $\tau = \{b_1, b_2, \overline{b_3}, \overline{b_4}, \overline{b_5}\}$. Since $\mathrm{O}(\tau) = 2$ the UB is then updated and search terminated.

The example combined with Proposition 6 implies the following.

▶ **Proposition 8.** *For every even $n \in \mathbb{N}$ there exists a PBO instance $\mathcal{F}_n$ on which the* `Extract-Cores` *subroutine of* `PBO-IHS` *extracts $\Omega(2^n)$ cores before termination if constraint seeding is not used and no cores if seeding is used.*

We observe an interesting connection between constraint seeding and *abstract cores*, a recently proposed improvement to the IHS algorithm for MaxSAT [6]. Abstract cores are a compact representation of a large number of ordinary core-constraints. More specifically, an abstraction variable $ab.c[k]$ defined over a set of n literals $ab \subset \mathrm{O}$ that all have the same coefficient in O has the definition $ab.c[k] \leftrightarrow \sum_{l \in ab} l \geq k$, i.e., the linear constraints $\sum_{l \in ab} l - k \cdot ab.c[k] \geq 0$ and $\sum_{l \in ab} l - n \cdot ab.c[k] < k$. Let Abs be a set of abstraction variables. An abstract core constraint C is a linear constraint for which $\textsc{var}(C) \subset \textsc{var}(\mathrm{O}) \cup Abs$ that is satisfied by any assignment that satisfies both $\mathcal{F}$ and the definitions of the abstraction variables. Each such constraint containing an abstraction variable $ab.c[k]$ corresponds to $\binom{n}{(n-k+1)}$ (non-abstract) core constraints of form $\sum_{l \in C, l \neq ab.c[k]} l + \sum_{l \in ab_k} l \geq 1$ where $ab_k \subset ab$ is any subset containing $n - k + 1$ variables.

A central motivation for abstract cores in the context of MaxSAT is that the IHS algorithm for MaxSAT needs to extract an exponential number of cores when solving the CNF translation of the instance presented in Example 2. As demonstrated by Example 7, the technique of constraint seeding in PBO already allows avoiding the need to extract a large number of core constraints on this specific instance.

4.2 Weight-Aware Core Extraction

Weight-aware core extraction (WCE), first proposed in the context of core-guided MaxSAT solving in [7], is a technique for extracting more core constraints from a single hitting set by using information provided by the coefficients of the objective variables. The idea has previously been explored in the context of PBO under the name independent cores in [21]. Here we employ WCE for the first time in the context of IHS.

Algorithm 3 details `Extract-Cores-WCE`, the computation of new core constraints with WCE. Given an instance $\mathcal{F}$ and a hitting set γ, the procedure initializes a weight $\mathcal{W}(l)$ for each objective function literal $l \in \mathrm{O}$. The weight of l equals its coefficient in O if $\gamma(l) = 0$ and 0 otherwise. Each call to `PB-Solve-A` is then performed with a set of assumptions containing all literals for which $\mathcal{W}(l)$ is not 0. Note specifically that the first set of assumptions will be same with and without employing WCE. After a subset κ of assumptions is obtained from the PB oracle, the weight of each literal $l \in \kappa$ is lowered by w^{κ}, the minimum weight among all literals in κ. Importantly, this lowers the weight of at least one literal to 0, thus guaranteeing theeventual termination of `Extract-Cores-WCE`.

The intuition underlying WCE is that it allows for extracting not only a (variable) disjoint set of core-constraints from each hitting set, but also core constraints whose variables intersect on a subset containing literals with large coefficients. The following example demonstrates how WCE can decrease the number of hitting sets that the IHS algorithm needs to compute before termination.

▶ **Example 9.** Consider an instance $\mathcal{F} = \{(b_1 + b_N \geq 1), (b_2 + b_N \geq 1), \ldots, (b_n + b_N \geq 1)\}$ with $\mathrm{O} = \sum_{i=1}^{n-1} b_1 + nb_N$. Invoke `Extract-Cores` (Algorithm 2) on $\mathcal{F}$ with $\gamma = \emptyset$. In the first iteration, `PB-Solve-A` is invoked with $\mathcal{A}_1 = \{b_1, \ldots, b_n, b_N\}$. Since any set κ that could be returned contains b_N it will be removed from the set of assumptions after one core has been computed. Since an assignment setting $b_N = 1$ satisfies the instance, `Extract-Cores` can only compute a single new core constraint before terminating. With WCE (i.e., `Extract-Cores-WCE`) the situation changes. The initial set of assumptions will again be $\mathcal{A}_1$. Since any set κ returned by `PB-Solve-A` will have $w^{\kappa} = 1$, the weight $\mathcal{W}$ of b_N is lowered by 1 and thus remains positive. Hence b_N will stay in the assumptions until either (i) n core-constraints have been extracted or (ii) all other literals are removed from the set of assumptions.

4.3 Non-Optimal Hitting Sets

At early stages of IHS search, when $\mathcal{C}$ only contains a few core constraints, we expect $\mathrm{O}(\gamma) < \mathrm{O}(\mathcal{F})$ to hold for an optimal hitting set γ over $\mathcal{C}$. Recalling that `PBO-IHS` can terminate only when $LB = \mathrm{O}(\mathcal{F})$, this implies that we do not expect an optimal hitting set over $\mathcal{C}$ to result in termination before enough cores have been extracted. However, the `Extract-Cores` subroutine does not necessarily need an *optimal* hitting set in order to compute new core constraints. Hence instead of spending time computing a—potentially useless—optimal hitting set, we can instead focus our efforts on computing any hitting set that allows `Extract-Cores` to derive more core constraints.

More precisely, we terminate `Min-Hs` once an incumbent hitting set γ^i is obtained which is either optimal or satisfies $\mathrm{O}(\gamma^i) < UB$. Even if the lower bound LB can only be updated if γ^i is optimal, `Extract-Cores` will still either derive a new core constraint, or find a solution τ for which $\mathrm{O}(\tau) = \mathrm{O}(\gamma^i) < UB$. In both cases, the search progresses toward an optimal solution. The only way in which γ^i can be rediscovered in subsequent iterations is if it was in fact optimal. More formally, we can show that the requirement of the hitting set being computed by `Min-Hs` either being optimal, or having cost lower than the current UB is sufficient for the correctness of `PBO-IHS`. This follows from the fact that each non-optimal hitting set can be computed by `Min-Hs` at most once and each optimal one at most twice.

For a more detailed argument in the context of MaxSAT, we refer the reader to [2].

4.4 Core Shrinking through Shuffling Assumptions

The sizes of core constraints found during iterations of IHS directly impact the tightness of the hitting set constraints. In IHS MaxSAT solving, subset-minimization of cores is done by iteratively asking the SAT solver performing cores extraction whether some soft clauses can be removed from the cores while maintaining unsatisfiability. However, in the context of PBO, we observed that subset-minimization of cores through the PB solver during IHS search often becomes too time-consuming, and hence we do not—at least currently—attempt to subset-minimize cores in this way before turning to the hitting set solver. Instead, we make use of another, computationally less demanding way of potentially identifying smaller cores. In particular, at the time of termination of the PB solver (the `PB-Solve-A` subroutine of Algorithm 2) at a specific iteration, the subset of assumptions from which the core constraint is formed is obtained by propagating all assumptions one by one until the solver reports unsatisfiable. A central fact to note is that the specific core constraint obtained will depend on the order in which assumptions are propagated; other orders of propagating the assumptions during this "analyzeFinal" phase may provide at times smaller cores. With this aim, we randomly shuffle the order of the assumptions a number of times (set to 20 repetitions in our current implementation), and choose a smallest-cardinality core among the cores obtained this way as the core constraint that is then added to the hitting set solver. Since this shuffling approach to shrinking cores relies only on polynomial-time propagation within the PB solver, it avoids the worst-case exponential subset-minimization calls if core shrinking would be performed by iteratively asking the PB solver to identify assumptions that can be left out from a found core.

4.5 Reduced Cost Fixing

The hybrid approach of `PBO-IHS` combining IP solving and PB reasoning opens up the possibility of introducing techniques from IP solving into the PB reasoning part of `PBO-IHS`. One such technique that we consider in this work is *reduced cost fixing*, a standard technique in the realm of IP solving [14, 15, 32]. In IHS for PBO, reduced cost fixing can be applied in two ways: on the LP relaxation of the actual PBO instance, and on the level of solving the hitting set programs using IP solving. In the context of IHS for MaxSAT and in particular on the level of the hitting set IP, reduced cost fixing was first explored in [2].

First consider employing reduced costs obtained from solving the hitting set problems during IHS search. For a set $\mathcal{C}$ of core-constraints and an objective function O, let $\texttt{Min-Hs}(\mathrm{O}, \mathcal{C})^{LP}$ be the LP relaxation of the IP depicted in Figure 1a, i.e., the linear program obtained by removing the requirement of the l variables being integral, and instead allowing them to take any value in the range $[0, 1]$. Informally speaking, given a solution η to $\texttt{Min-Hs}(\mathrm{O}, \mathcal{C})^{LP}$, the reduced cost $rc(b)$ of a variable assigned to 1 (0) by η measures the effect that assigning b to 0 (1) instead would have on $\mathrm{O}(\eta)$. Since the optimal cost of $\texttt{Min-Hs}(\mathrm{O}, \mathcal{C})^{LP}$ is a lower bound on the optimal cost $\texttt{Min-Hs}(\mathrm{O}, \mathcal{C})$ which in turn is a lower bound on $\mathrm{O}(\mathcal{F})$, the reduced costs of a variable b in the objective function can sometimes be used to show that either $b = 1$ or $b = 0$ holds for at least one optimal solution to $\mathcal{F}$, which allows us to fix the value of b for the rest of the search.

More precisely, suppose τ_{best} is a feasible solution to $\mathcal{F}$ and consider non-basic variable x (i.e., a variable assigned to either 0 or 1 by η) of $\texttt{Min-Hs}(\mathrm{O}, \mathcal{C})^{LP}$. If $\eta(x) = 0$ and either: (i) $\mathrm{O}(\eta) + rc(x) > \mathrm{O}(\tau_{best})$ or (ii) $\mathrm{O}(\eta) + rc(x) = \mathrm{O}(\tau_{best})$ and $\tau_{best}(x) = 0$, then x is fixed to

0 in subsequent iterations of the PBO-IHS algorithm. Similarly, if $\eta(x) = 1$ and either: (i) $\mathrm{O}(\eta) - rc(x) > \mathrm{O}(\tau_{best})$ or (ii) $\mathrm{O}(\eta) - rc(x) = \mathrm{O}(\tau_{best})$ and $\tau_{best}(x) = 1$, then x is fixed to 1 is subsequent iterations. We emphasise, that in both cases, the variable is fixed both in the Min-Hs, and the Extract-Cores subroutines.

A detailed argument for the correctness of reduced cost fixing in implicit hitting set-based MaxSAT can be found in [2]. We sketch the proof of the case $\eta(x) = 0$. First note that if $x = 1$ is infeasible for the LP relaxation of Min-Hs, then it will be infeasible for the IP as well. In other words, then no hitting set over $\mathcal{C}$ can set $x = 1$ and, by the definition of a core constraint, neither can any solution to $\mathcal{F}$. On the other hand, if $x = 1$ is feasible, then by the properties of reduced costs [4], any solution η^m to the LP for which $\eta^m(x) = 1$ will have $\mathrm{O}(\eta^m) \geq \mathrm{O}(\eta) + rc(x) \geq \mathrm{O}(\tau_{best}) \geq \mathrm{O}(\mathcal{F})$. Since the LP is a relaxation of the IP and the costs of the optimal solutions γ^o of the IP have $\mathrm{O}(\gamma^o) \leq \mathrm{O}(\mathcal{F})$, it follows that fixing $x = 0$ can be done without removing an optimal solution of the IP.

Secondly, we note that the LP relaxation of the input PBO instance itself can be solved for obtaining bounds information already before the IHS search, complementary to the information obtained from reduced costs from the hitting set computations during search. In particular, for obtaining reduced costs information on an input PBO instance $\mathcal{F}$, solve the LP relaxation $\mathcal{F}^{LP}$ of $\mathcal{F}$ prior to starting the main search loop, and apply reduced cost fixing based on the reduced costs obtained from an optimal solution η^i of $\mathcal{F}^{LP}$ whenever the IHS search improves the upper bound UB during search.

5 Empirical Evaluation

We turn to overviewing results from an empirical evaluation of the IHS approach to PBO presented in this work. The experiments reported on were run on nodes with 8-core Intel Xeon E5-2670 2.6-GHz CPUs and 64-GB RAM. We set a per-instance 3600-second time and 16-GB memory limit.

5.1 Implementation

We implemented PBO-IHS in Python, with a PB solver (as the core extractor) and an integer programming solver (as the hitting set solver) imported as external modules. We use the Roundingsat version 2 [24] (commit 1476bf0bcd) as the PB solver, using its most recent configuration in [20]. To implement the PB-Solve-A function, we extended the Roundingsat implementation to include an analyzeFinal function similar to the one implemented in the MiniSat SAT solver [22, 23], so that we can call Roundingsat within PBO-IHS under assumptions and extract unsatisfiable cores over the assumptions. As the integer programming solver for hitting set computations we used IBM ILOG CPLEX C++ API version 12.8. We compiled both Roundingsat and CPLEX API components using pybind11, which is a utility that allows to compile C++ libraries as python modules. In the following, we will refer as PBO-IHS to our implementation of the IHS approach to PBO which applies HS reduced cost fixing, constraint seeding, assumption set shuffling, non-optimal hitting sets and weight-aware core extraction, but does not apply reduced cost fixing based on solving the LP relaxation of the input PBO instance and does not employ abstract cores. (To this end, we will also report on the marginal contribution of each of these search techniques on the overall performance of PBO-IHS.) For the experiments, our implementation of PBO-IHS runs single-threadedly. The PBO-IHS implementation is available in open source at https://bitbucket.org/coreo-group/pbo-ihs-solver/.

5.2 Alternative Approaches

We extensively compare the empirical performance of `PBO-IHS` to those of previously proposed specialized approaches to PBO:

- **Open-WBO** [30] encodes the PBO instance into a MaxSAT instance by transforming the PB constraints into CNF by the well-known (generalized) totalizer encoding [29]. The MaxSAT instance is then solved with the OLL algorithm for MaxSAT [31].
- **Sat4J** [8] generalizes the CDCL procedure for SAT solving to PB solving and the cutting planes proof system. The cutting planes reasoning is implemented using the weakening and saturation rules similar to [10]. Computing an optimal solution to an instance $\mathcal{F}$ is done by *solution improving search*, i.e., starting from $ub = \infty$ iteratively invoking the solver on the formula $\mathcal{F} \cup \{\sum_{(w,l) \in \mathrm{O}} wl < ub\}$ which is satisfiable by an assignment τ iff τ is a solution to $\mathcal{F}$ with $\mathrm{O}(\tau) < ub$. When such τ is found, ub is updated to $\mathrm{O}(\tau)$ and the loop reiterated. The search terminates when the solver reports the formula to be unsatisfiable, at which point the last found (optimal) solution is returned.
- **NaPS** [40] encodes the PBO instance into a MaxSAT instance using binary decision diagrams (BDDs). An optimal solution to the MaxSAT instance is then computed a combination of solution improving and binary search.
- **Roundingsat (RS)** [24] generalizes the CDCL procedure for SAT solving to PB solving and the cutting planes proof system. Cutting planes reasoning is implemented using the division and rounding rules. Optimization is then done by solution improving search.
- **RS/lp** [20], a version of RS that periodically invokes a linear programming (LP) solver on the LP relaxation of the instance being solved. The LP calls are used to derive more conflicts to the CDCL procedure implemented in basic roundingsat. For example, if there are no feasible solutions to the LP relaxation of the instance under the current partial assignment, then there will not be any feasible solutions to the PB instance either. Computing an optimal solution is done by solution improving search.
- **RS/oll** [21], a version of RS that combines the solution improving search with an extension of the OLL algorithm to PBO [1]. OLL is a lower bounding approach that extracts core constraints of the instance being solved. Based on the obtained constraints, the instance is then relaxed in a way that allows —in a controlled way— one more of the literals in the objective function to be set to 1 in subsequent iterations.

In addition to these academic specialized PBO solvers, we also investigate how `PBO-IHS` fares against **CPLEX** [13].

5.3 Benchmarks

For the experiments, we collected a large number of benchmarks from different sources. Firstly, we collected all benchmarks used in Pseudo-Boolean Competition 2016 [35] (so far the most recent instantiation of the competition) as well as benchmarks available on the competition website that were used in previous competition instantiations since 2005. Secondly, we collected all 0-1 integer programs from the MIPLIB 2017 library [26] as well as earlier MIPLIB releases. We filtered out 7914 benchmark instances that had no objective function and 249 unsatisfiable benchmarks which do not admit solutions, as uninteresting for benchmarking optimization solvers, as well as 206 benchmarks that have at least one coefficient with an absolute value higher than 2^{64} and 548 benchmarks with non-linear constraints or non-linear terms in their objective functions. Starting from 17312 Pseudo-Boolean Competiton benchmarks and 1273 MIPLIB benchmarks, respectively, after filtering we were left with 8456 and 252 benchmarks, respectively, giving a total of 8708 benchmarks that we used in the experiments reported on here.

We categorized to the best of our knowledge the benchmarks (based on their source, related publications, and finally, by file names) into different problem domains, obtaining the problem domain categorization shown in Table 1. We observe that the whole benchmark set is significantly unbalanced in terms of the number of instances originating from specific problem domains. 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 30 instances (or all of the instances from the domain, if the domain included less that 30 instances) for the comparison. The sampled benchmark set contains in total 1759 benchmarks. Unless explicitly stated otherwise, all results reported on in this section are with respect to the sampled benchmark set.

5.4 Results: Comparison with Specialized PBO Solvers

We first compare the empirical performance of `PBO-IHS` to those of other specialized PBO solvers on the sampled benchmark set. Figure 2(top) shows how many benchmarks each solver was able to solve (y-axis) under different per-instance time limits, 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. To justify the sampling of benchmarks in order to achieve a balanced benchmark set, confirmed the results for the three best-performing solvers under 10 different random samplings. The results, shown in Figure 2(bottom), confirm that the relative performance of the solvers is robust against subsampling benchmarks in a balanced way. In more detail, For each solver S, Figure 2(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.

More detailed data per benchmark domain (over the *full* benchmark set) is reported in Table 1, with the number of instances solved (left column) and the cumulative runtimes over solved instances (right column) shown for each solver, with all benchmarks from each problem domain included. Interestingly, we observe that the relative performance of the Roundingsat versions (RS/lp and RS/oll) and `PBO-IHS` depends significantly on the problem domain, suggesting that the approaches complement each other.

5.5 Results: Impact of Different Search Techniques in PBO-IHS

We also investigated the marginal impact of the different search techniques and refinements to `PBO-IHS` on the empirical performance of `PBO-IHS`. Figure 3(top) provides a comparison of the default configuration of `PBO-IHS` (with HS reduced cost fixing (hs-rc), constraint seeding, assumption set shuffling, non-optimal hitting sets, weight-aware core extraction, but without reduced cost fixing based on solving the LP relaxation of the input PBO instance (pb-rc) or abstract cores) to configurations of `PBO-IHS` with each of HS reduced cost fixing, constraint seeding, assumption set shuffling, non-optimal hitting set computation, and weight-aware core extraction separately switched off, as well the configurations using reduced cost fixing on the PBO LP and abstract cores separately. We observe that constraint seeding makes the largest positive marginal contribution to the empirical performance of `PBO-IHS`, and assumption set shuffli 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. The use of abstract cores, at least as currently implemented, makes a significant negative marginal contribution, noticeably degrading the performance of the default version of PBO-IHS. Exploring the relationship between constraint seeding and abstract cores in PBO, as well as alternative instantiations of the abstract cores framework, remains interesting for future work. The two different forms of reduced cost have only a very modest impact. While reduced cost fixing based on the PBO LP does not make a significant negative marginal contribution, it does not appear to improve on the performance of PBO-IHS, which justifies disabling it together with abstract cores in the default configuration of PBO-IHS.

5.6 Results: Runtime Division between Core Extraction and MCHS

Figure 4(left) details the fraction of solving time spent in the Min-Hs subroutine of PBO-IHS on the 898 of the instances solved within the time limit. Note that since the Min-Hs and

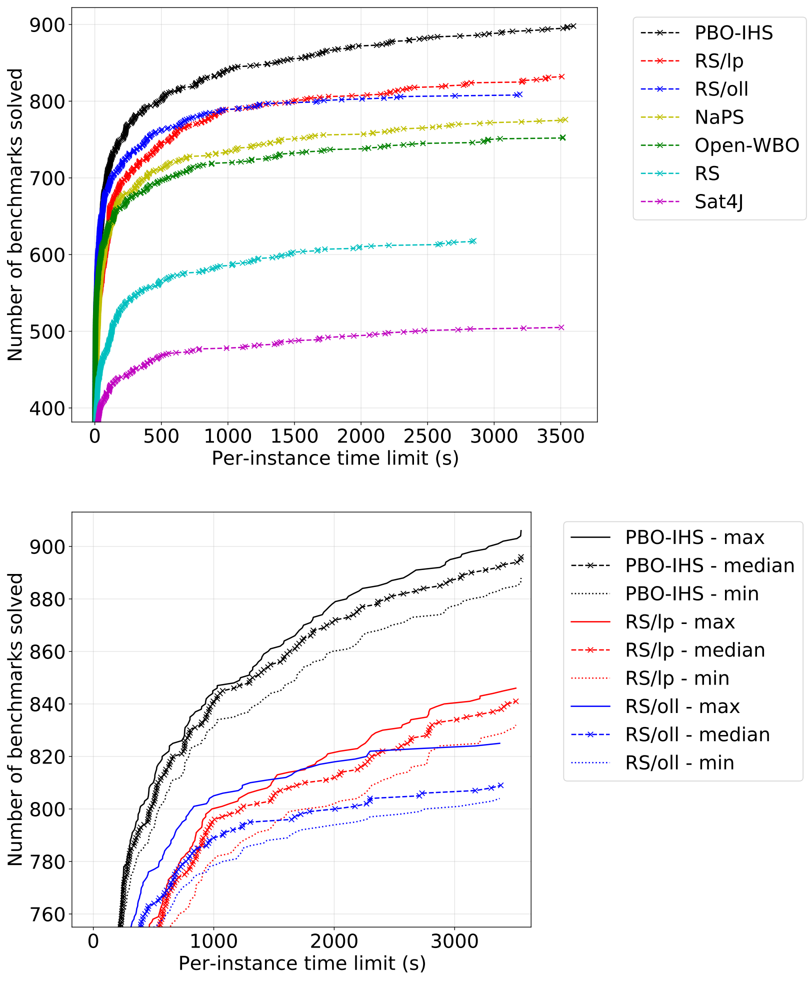

Figure 2 Top: Runtime comparison of specialized PBO solvers. Bottom: Confidence intervals over 10 benchmark subset samples for the three best-performing solvers.

Table 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

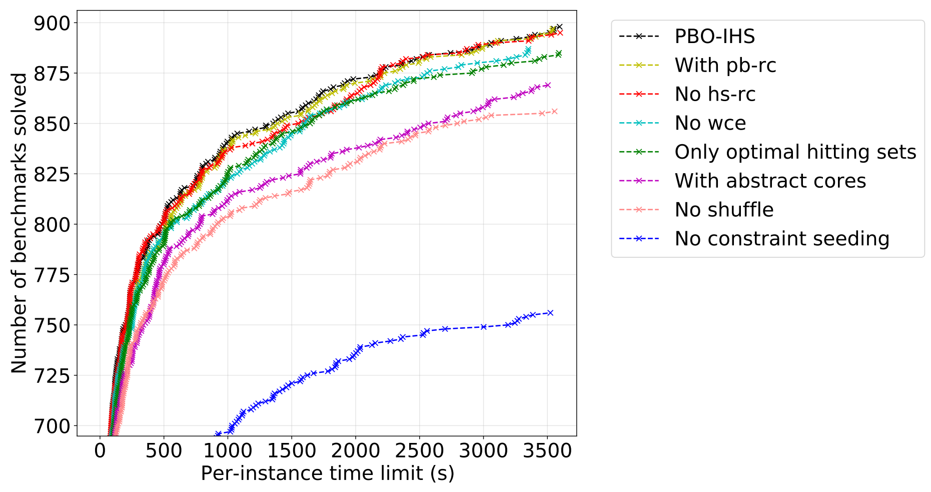

Figure 3 Runtime comparison of various `PBO-IHS` variants.

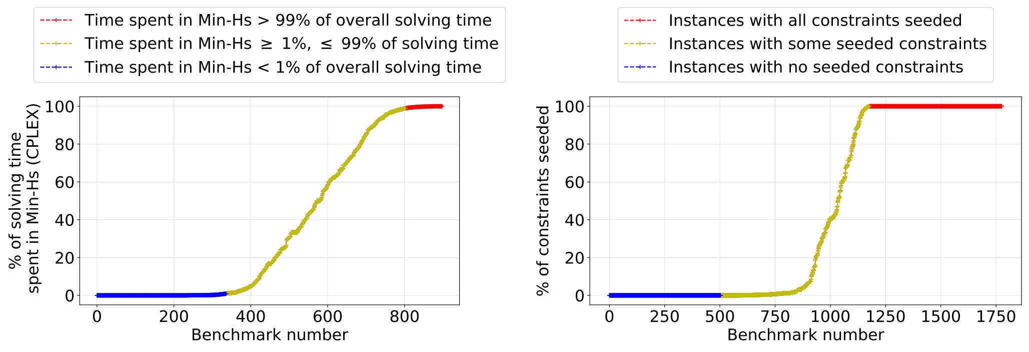

Figure 4 Left: Ratio of solving time spent by `PBO-IHS` in `Min-Hs` subroutine for solved benchmarks. Right: Ratio of constraints seeded on all benchmarks.

`PB-Solve-A` subroutines dominate the running time of `PBO-IHS`, the rest of the runtime is effectively spent in `PB-Solve-A`. We observe that on most of the instances, over 80% of the overall solving time is spent computing core constraints: on 462 of the 893 instances, only 20% of the time was spent in `Min-Hs`, and one over 1/3 of the instances 99% of the overall solving time is spent in `PB-Solve-A` (marked by the blue line). On the other hand, the runtimes of `Min-Hs` dominates on approximately 1/5 of the instances.

Figure 4(right) shows the fractons of constraints that can be seeded over all benchmark instances. At least one constraint is seeded for 71.4% of the instancess; 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 while the whole instance is solved directly as an IP through a single `Min-Hs` call when all constraints are seeded, we also observed that there are instances on which the runtime of `Min-Hs` dominates even though all constraints are not seeded.

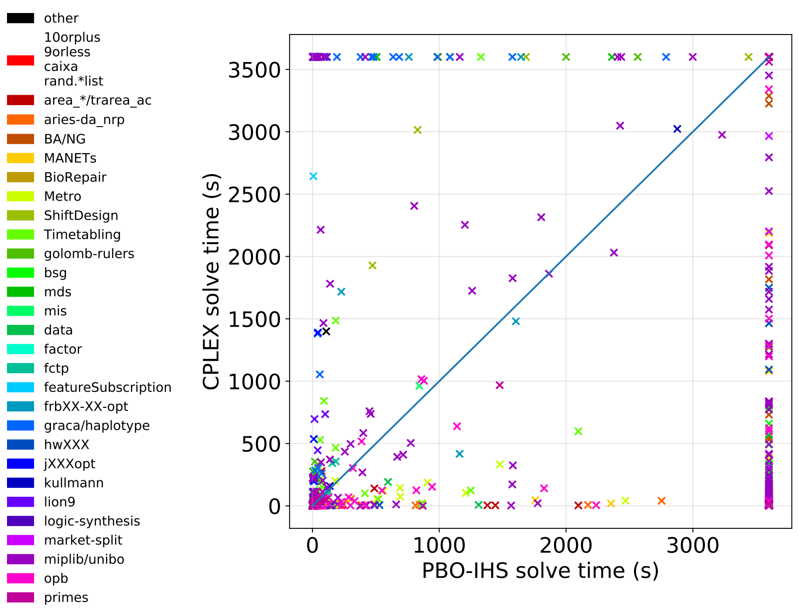

Figure 5 Per-instance runtime comparison of `PBO-IHS` (x-axis) vs CPLEX (y-axis).

5.7 Results: Comparison with a Commercial IP Solver

Finally, we investigate how the prototype implementation of `PBO-IHS` fares in terms of runtime performance against CPLEX, one of the de-facto commercial MIP solvers with a significant number of person years behind it. For a fair comparison with CPLEX, we used the CPLEX presolver also before calling `PBO-IHS`. 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 calling `PBO-IHS`). A per-instance runtime comparison is shown in Figure 5, with more details per benchmark domain provided in Appendix A. We observe that, while CPLEX fairs better in the overall number of solved instances, the two solvers exhibit noticeably complementary performance, relative performance depending on the problem domain considered.

6 Conclusions

We described and implemented a first instantiation of the implicit hitting set approach for pseudo-Boolean optimization. On one hand, the instantiation is motivated by the great success of the implicit hitting set approach in the context of maximum satisfiability, which motivates extending the approach to the more generic context of PBO. On the other hand, the instantiation is motivated by recent advances in pseudo-Boolean solving as a generalization of SAT solving, providing efficient unsatisfiable core extraction which is one of the critical requirements for realizing IHS for PBO. We studied the impact of liftings of various IHS search techniques from MaxSAT to PBO, and showed through an extensive empirical evaluation that our IHS PBO solver implementation provides in practice a competitive as well as complementary approach to pseudo-Boolean optimization.

References

1 Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in CLASP. In Agostino Dovier and Vítor Santos 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. doi:10.4230/LIPIcs.ICLP.2012.211.

2 Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in MaxSAT. In Christopher 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. doi:10.1007/978-3-319-66158-2_41.

3 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. *Maximum Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 24, pages 929–991. IOS Press BV, 2021. doi:10.3233/FAIA201008.

4 Omid Sanei Bajgiran, André A. Ciré, and Louis-Martin Rousseau. A first look at picking dual variables for maximizing reduced cost fixing. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*, volume 10335 of *Lecture Notes in Computer Science*, pages 221–228. Springer, 2017. doi:10.1007/978-3-319-59776-8_18.

5 Peter Barth. Linear 0-1 inequalities and extended clauses. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning,4th International Conference, LPAR'93, St. Petersburg, Russia, July 13-20, 1993, Proceedings*, volume 698 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1993. doi:10.1007/3-540-56944-8_40.

6 Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set MaxSat solving. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020. doi:10.1007/978-3-030-51825-7_20.

7 Jeremias Berg and Matti Järvisalo. Weight-aware core extraction in SAT-based MaxSAT solving. In J. Christopher 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. doi:10.1007/978-3-319-66158-2_42.

8 Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–6, 2010. doi:10.3233/sat190075.

9 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans Van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 133–182. IOS Press BV, 2021. doi:10.3233/FAIA200990.

10 Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, pages 830–835. ACM, 2003. doi:10.1145/775832.776041.

11 Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. A modular approach to MaxSAT modulo theories. In Matti Järvisalo and Allen Van 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. doi:10.1007/978-3-642-39071-5_12.

12 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987. doi:10.1016/0166-218X(87)90039-4.

13 IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

14 Harlan P. Crowder, Ellis L. Johnson, and Manfred W. Padberg. Solving large-scale zero-one linear programming problems. *Operational Research*, 31(5):803–834, 1983. `doi:10.1287/opre.31.5.803`.

15 George B. Dantzig, D. Ray Fulkerson, and Selmer M. Johnson. Solution of a large-scale traveling-salesman problem. *Operational Research*, 2(4):393–410, 1954. `doi:10.1287/opre.2.4.393`.

16 Jessica Davies. *Solving MaxSAT by decoupling optimization and satisfaction.* PhD thesis, University of Toronto, 2013.

17 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man 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. `doi:10.1007/978-3-642-23786-7_19`.

18 Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian 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. `doi:10.1007/978-3-642-40627-0_21`.

19 Erin Delisle and Fahiem Bacchus. Solving weighted CSPs by successive relaxations. In Christian 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 273–281. Springer, 2013. `doi:10.1007/978-3-642-40627-0_23`.

20 Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 2021. `doi:10.1007/s10601-020-09318-x`.

21 Jo Devriendt, Stephan Gocht, Emir Demirovic, Jakob Nordström, and Peter 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`.

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

23 Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003. `doi:10.1016/S1571-0661(05)82542-3`.

24 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In Jérôme 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. `doi:10.24963/ijcai.2018/180`.

25 Katalin Fazekas, Fahiem Bacchus, and Armin Biere. Implicit hitting set algorithms for maximum satisfiability modulo theories. In Didier Galmiche, Stephan Schulz, and Roberto 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. `doi:10.1007/978-3-319-94205-6_10`.

26 Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 2021. `doi:10.1007/s12532-020-00194-3`.

27 Antti Hyttinen, Paul Saikko, and Matti Järvisalo. A core-guided approach to learning optimal causal graphs. In Carles 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. `doi:10.24963/ijcai.2017/90`.

28 Alexey Ignatiev, Alessandro Previti, Mark H. Liffiton, and João Marques-Silva. Smallest MUS extraction with minimal hitting set dualization. In Gilles 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. `doi:10.1007/978-3-319-23219-5_13`.

29 Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-Boolean constraints. In Gilles 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. `doi:10.1007/978-3-319-23219-5_15`.

30 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In Carsten Sinz and Uwe 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. `doi:10.1007/978-3-319-09284-3_33`.

31 António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014. `doi:10.1007/978-3-319-10428-7_41`.

32 George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Interscience Series in Discrete Mathematics and Optimization. Wiley, 1988. `doi:10.1002/9781118627372`.

33 Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991. `doi:10.1137/1033004`.

34 Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006. URL: `https://www.sciencedirect.com/science/bookseries/15746526/2`.

35 Olivier Roussel. Pseudo-Boolean competition 2016. `http://www.cril.univ-artois.fr/PB16/`. Accessed: 25 April 2021.

36 Olivier Roussel and Vasco Manquinho. Pseudo-boolean and cardinality constraints. In Armin Biere, Marijn Heule, Hans Van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, chapter 28, pages 1087–1129. IOS Press BV, 2021. `doi:10.3233/FAIA201012`.

37 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In Nadia Creignou and Daniel 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. `doi:10.1007/978-3-319-40970-2_34`.

38 Paul Saikko, Carmine Dodaro, Mario Alviano, and Matti Järvisalo. A hybrid approach to optimization in answer set programming. In Michael Thielscher, Francesca Toni, and Frank 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`.

39 Paul Saikko, Johannes Peter Wallner, and Matti Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In Chitta Baral, James P. Delgrande, and Frank 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`.

40 Masahiko Sakai and Hidetomo 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. `doi:10.1587/transinf.2014FOP0007`.

41 Roberto Sebastiani and Patrick Trentin. OptiMathSAT: A tool for optimization modulo theories. *Journal of Automated Reasoning*, 64(3):423–460, 2020. `doi:10.1007/s10817-018-09508-6`.

42 Hossein M. Sheini and Karem A. Sakallah. Pueblo: A hybrid pseudo-Boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):165–189, 2006. `doi:10.3233/sat190020`.

A Detailed Results: PBO-IHS vs CPLEX

Table 2 provides a per-instance comparison of the performance of PBO-IHS and CPLEX. on the *full* benchmark set.

Table 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