Improvements to the Implicit Hitting Set Approach to Pseudo-Boolean Optimization

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

The development of practical approaches to efficiently reasoning over pseudo-Boolean constraints has recently increasing attention as a natural generalization of Boolean satisfiability (SAT) solving. Analogously, solvers for pseudo-Boolean optimization draw inspiration from techniques developed for maximum satisfiability (MaxSAT) solving. Recently, the first practical solver lifting the implicit hitting set (IHS) approach—one of the most efficient approaches in modern MaxSAT solving—to the realm of PBO was developed, employing a PB solver as a core extractor together with an integer programming solver as a hitting set solver. In this work, we make practical improvements to the IHS approach to PBO. We propose the integration of solution-improving search to the PBO-IHS approach, resulting in a hybrid approach to PBO which makes use of both types of search towards an optimal solution. Furthermore, we explore the potential of different variants of core extraction within PBO-IHS—including recent advances in PB core extraction, allowing for extracting more general PB constraints compared to the at-least-one constraints typically relied on in IHS—in speeding up PBO-IHS search. We show that the empirical efficiency of PBO-IHS—recently shown to outperform other specialized PBO solvers—is further improved by the integration of these 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, solution-improving search, unsatisfiable cores

Digital Object Identifier 10.4230/LIPIcs.SAT.2022.13

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. The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

1 Introduction

Pseudo-Boolean (PB) constraints, i.e., linear inequalities over binary variables with integer coefficients, offer a natural approach to modeling various types of NP-hard real-world problems [5]. Viewed as natural generalizations of conjunctive normal form clauses, liftings of conflict-driven CDCL search [34, 26] for deciding the satisfiability of a given set of PB constraints have been developed [18]. This line of work is motivated further by the fact that natively reasoning on the level of PB constraints allows for implementing proof systems that are stronger than the resolution proof system underlying CDCL [18, 6, 33, 7]. Furthermore, generalizing conflict-driven search to the realm of PB allows for PB solving under assumptions and thereby the extraction of unsatisfiable cores (inconsistent assignments

over literals of interest) on the PB-level [14]. Practical approaches to unsatisfiable core extraction for sets of pseudo-Boolean constraints allow for a variety of possible ways of going from deciding satisfiability to pseudo-Boolean optimization (PBO) [6, 33, 4, 32]. In particular, recent developments in lifting ideas from modern maximum satisfiability (MaxSAT) solving to PBO—also known as 0-1 integer programming—have proven to yield effective solving approaches to PBO as alternatives to classical integer programming solving techniques. As generalizations of some of the most successful MaxSAT solving approaches, these recent developments include both core-guided PBO solvers [14] and, most recent, instantiations of the so-called implicit hitting set approach (IHS) in the realm of PBO [35].

In this work, we focus on the recently-proposed implicit hitting set approach to PBO, implemented in the PBO-IHS solver [35]. The implicit hitting set approach is a relatively generic solving paradigm that has seen successful practical instantiations in various settings concerning finding optimal solutions to NP-hard optimization problems [11, 12, 23, 31, 22, 19, 30, 24]. The IHS approach is an iterative lower-bounding approach at heart, based on accumulating unsatisfiable cores of a problem declaration (using an unsatisfiability core extractor for the declaration language at hand) and finding hitting sets over the accumulated set of unsatisfiable cores (typically using an integer programming solver), until ruling out a most-recent optimal hitting set (using assumptions) yields a solution satisfying the rest of the constraints. As shown in [35], the recent PBO-IHS solver implementing an IHS approach to PBO has proven empirically effective, surpassing other specialized approaches to PBO—including recent developments in core-guided PBO solving—in efficiency and at times surpassing even the efficiency of commercial integer programming systems.

Motivated by these recent developments, in this work we study possibilities of further improving the performance of PBO-IHS through integrating additional search techniques to the solver. In particular, firstly, we propose the integration of solution-improving search to the PBO-IHS approach, resulting in a hybrid approach to PBO making use of both types of search towards an optimal solution. In itself, solution-improving search [4, 32, 13], also known as model-improving search in the realm of MaxSAT [28, 2], is an upper-bounding approach that finds better solutions using a decision oracle until the currently known best solution is proven to be optimal. Integrating solution-improving search heuristically within the PBO-IHS search loop gives a guarantee that the current upper bound solution is improved. The improved bound can subsequently be made use of by IHS search techniques. Secondly, we explore the potential of variants of core extraction within PBO-IHS for improving the efficiency of PBO-IHS. Whereas the earlier version of PBO-IHS made use of MiniSat-style conflict analysis, resulting in SAT-like unsatisfiable cores—specifically, at-least-one cardinality constraints—we study the impact of making use of recent advances in PB core extraction which allow for natively extracting unsatisfiable cores expressed as more general PB constraints [14], as well as ways of making use of both types of cores. Furthermore, we study the impact of obtaining multiple cores per search iteration through lightweight assumption shuffling on the runtime efficiency of PBO-IHS. As an end-result, we provide a performance-boosted PBO-IHS solver with state-of-the-art performance on a wide range of standard benchmarks.

2 Preliminaries

A binary variable x has the domain $\{0, 1\}$. A literal l over a variable x is either x or $\overline{x} \equiv (1-x)$, in both cases the variable $\text{VAR}(l) = x$ of l is x. A pseudo-Boolean (PB) constraint (or simply constraint, for short) C is a 0-1 integer linear inequality $\sum_i a_i l_i \geq B$ over literals l_i, with the bound B and each coefficient a_i integer constants. The set of variables appearing in C is

denoted by $\text{VAR}(C)$. We assume w.l.o.g. that each constraint C is in normalized form, i.e., that each variable appearing in C is distinct and that the coefficients a_i and bound B are non-negative. An assignment $\tau\colon \text{VAR}(C) \to \{0,1\}$ is extended to literals by $\tau(\bar{l}) = 1 - \tau(l)$. Such an assignment τ satisfies C (denoted by $\tau(C) = 1$) if $\sum_i a_i \tau(l_i) \geq B$ and falsifies C otherwise. When convenient we view an assignment τ over a set X of variables as the 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 $\text{VAR}(F)$ the set of variables appearing in the constraints of F. An assignment $\tau\colon \text{VAR}(F) \to \{0,1\}$ is a solution to F if it satisfies each 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 $\mathcal{I} = (F, \text{O})$ of the pseudo-Boolean optimization problem (PBO) consists of a PB formula F and an objective function $\text{O} \equiv \sum_i w_i l_i$ where each l_i is a literal over a variable $\text{VAR}(l_i) \in \text{VAR}(F)$ and w_i its non-negative integer coefficient. We will sometimes abuse notation and treat O as either a set of literals or a set of coefficient-literal tuples, i.e., write $l \in \text{O}$ and $(w, l) \in \text{O}$ to denote that O contains a literal l or the term wl. The set of variables appearing in O is denoted by $\text{VAR}(\text{O})$. The value of O under an assignment $\tau\colon \text{VAR}(\text{O}) \to \{0,1\}$ is $\text{O}(\tau) = \sum_i w_i \tau(l_i)$. The set of solutions to $\mathcal{I}$ consists of the assignments that are solutions to F. A solution τ is optimal if it minimizes $\text{O}(\tau)$ over all solutions to $\mathcal{I}$. The cost of the optimal solutions of $\mathcal{I}$ is denoted by $\text{O}(\mathcal{I})$. The PBO problem asks to find an optimal solution to a given PBO instance.

The algorithm for computing optimal solutions to PBO instances that we focus on makes use of so-called *core constraints* and *hitting sets*. A constraint C is a core constraint of a PBO instance $\mathcal{I} = (F, \text{O})$ if (i) the variables of C consist only of variables in O (i.e., $\text{VAR}(C) \subset \text{VAR}(\text{O})$) and (ii) all solutions of $\mathcal{I}$ satisfy C.

▶ **Example 1.** Consider the PBO instance $\mathcal{I} = (F, \text{O})$ with $F = \{a_1 + (1 - x_1) \geq 1, a_2 + (1 - x_2) \geq 1, x_1 + x_3 \geq 1, x_2 + (1 - x_3) \geq 2\}$ and $\text{O} \equiv a_1 + a_2$. The solution $\tau = \{a_1, a_2, x_1, x_2, \overline{x_3}\}$ is an example of an optimal solution to $\mathcal{I}$, and has cost $\text{O}(\tau) = \text{O}(\mathcal{I}) = 2$. The constraints $a_1 + a_2 \geq 1$ and $a_1 + a_2 \geq 2$ are examples of core constraints of $\mathcal{I}$. This can be seen by verifying that no solution τ_s of F assigns $\tau_s(a_1) = \tau_s(a_2) = 0$.

Given a set $\mathcal{C}$ of core constraints of a PBO instance, we say that an assignment $\gamma\colon \text{VAR}(\text{O}) \to \{0,1\}$ that satisfies $\mathcal{C}$ is a *hitting set* of $\mathcal{C}$. A hitting set γ^o is optimal if $\text{O}(\gamma^o) \leq \text{O}(\gamma)$ holds for each hitting set γ of $\mathcal{C}$. The term hitting set stems from an important special case of core constraints, namely, those of form $C = \sum l \geq 1$. A set of such constraints can be viewed as an instance of the classical hitting set problem: each such constraint is satisfied by setting at least one of the literals in C to 1, thus *hitting* that constraint. The `PBO-IHS` algorithm we focus on in this work makes use of the well-known fact that hitting sets provide lower bounds on $\text{O}(\mathcal{I})$, stated formally as follows.

▶ **Proposition 2** (See e.g. [35]). *Let γ^o, $\mathcal{C}$ and $\mathcal{I}$ be as above. Then $O(\gamma^o) \leq O(\mathcal{I})$.*

3 The PBO-IHS Implicit Hitting Set Approach to PBO

In this work, we study ways of improving the performance of the recently-proposed PBO-IHS implicit hitting set approach to pseudo-Boolean optimization [35].

Algorithm 1 details the `PBO-IHS` algorithm for computing an optimal solution to a PBO instance $\mathcal{I}$ given as input. The algorithm works by iteratively refining an upper bound UB and a lower bound LB on the optimal cost $\text{O}(\mathcal{I})$ of $\mathcal{I}$. The algorithm also maintains a witness for the upper bound UB in the form of an assignment τ_{best} for which $\text{O}(\tau_{best}) = UB$.

```
1  PBO-IHS(I)
       Input: A PBO instance I = (F, O)
       Output: An optimal solution τ
2      (τ_best, sat?) ← PB-Solve(F);
3      if not sat? then
4          return "no feasible solutions";
5      UB ← O(τ_best); LB ← 0;
6      C ← SeedConstraints(F, VAR(O));
7      while TRUE do
8          (γ, opt?) ← ComputeHittingSet(O, UB, C);
9          if opt? then LB ← O(γ);
10         if UB = LB then break ;
11         C ← C ∪ ReducedCostFixing(O, τ_best, C);
12         C ← C ∪ ExtractCores(γ, UB, τ_best, I);
13         if UB = LB then break;
14     return τ_best
```

Algorithm 1 The PBO-IHS algorithm 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$\}$

Figure 1 IP for computing an optimal hitting set over core constraints

```
1  ExtractCores(γ, UB, τ_best, I = (F, O))
2      C_n ← ∅; W ← {(x, w) | x = VAR(l) ∧ (l, w) ∈ O};
3      while TRUE do
4          γ_A ← {l | l ∈ γ ∧ W(l) > 0};
5          (sat?, K, τ) ← PB-SolveUnderAssumptions(F, γ_A);
6          if (sat?) then
7              if O(τ) < UB then τ_best ← τ; UB ← O(τ);
8              return C_n;
9          else
10             C_n ← C_n ∪ K;
11             for C ∈ K do
12                 w^C = min_{x∈VAR(C)}{W(x)};
13                 for x ∈ VAR(C) do W(x) ← W(x) − w^C;
```

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

The search terminates when $LB = UB$ at which point τ_{best} is returned as a provably-optimal solution.

In more detail, when invoked on a PBO instance $\mathcal{I} = (F, \mathrm{O})$, a PB solver is invoked on F (Line 2). If F is found to be unsatisfiable, there are no solutions to the PBO instance and the search terminates (Line 4). If F is satisfiable, the PB solver returns a solution τ_{best} of $\mathcal{I}$. The upper bound UB on $\mathrm{O}(\mathcal{I})$ is initialized to $\mathrm{O}(\tau_{best})$ and the lower bound LB is initialized to 0 (Line 5). The set $\mathcal{C}$ of so-far accumulated core constraints of $\mathcal{I}$ is initialized to consist of all constraints C of F for which $\mathrm{VAR}(C) \subset \mathrm{VAR}(\mathrm{O})$, i.e., all constraints that only contain literals over variables in the objective function (Line 6; this is the so-called constraint seeding step, for more see [35]).

The main search loop of PBO-IHS consists of Lines 7-13. Each iteration of the main search loop starts with the computation of a hitting set over $\mathcal{C}$ (Line 8). In more detail, the procedure ComputeHittingSet computes a hitting set γ of $\mathcal{C}$ that is either optimal or has cost lower than the current upper bound UB. Such γ is computed by invoking an integer

programming solver on the integer program `Min-Hs`$(\mathrm{O}, \mathcal{C})$ (detailed in Figure 1) representing the hitting set problem over $\mathcal{C}$ with O as the objective function. The integer programming solver is ran until an incumbent solution is reached that is either optimal or has cost lower than UB. The procedure returns the hitting set γ as well as a boolean *opt?* indicating whether γ is an optimal hitting set of $\mathcal{C}$. If γ is optimal, then $\mathrm{O}(\gamma) \leq \mathrm{O}(\mathcal{I})$ by Proposition 2. Hence in this case the lower bound LB is updated (Line 9). If the upper and lower bounds match, the search terminates at Line 10.

In case the upper and lower bounds do not match at this point, the `ReducedCostFixing` procedure is invoked in an attempt to fix the values of yet-not-fixed variables in $\mathcal{I}$ (and thereby also in `Min-Hs`$(\mathrm{O}, \mathcal{C})$) via the reduced cost fixing technique [9, 10, 27], similarly as employed in the context of IHS for maximum satisfiability [1], based on the current bounds information and so-called reduced costs obtained without extract overhead from the latest IP solver invocation (Line 11). The iteration ends with a core extraction step through `ExtractCores` (Line 12) using a PB solver to extract core constraints of $\mathcal{I}$ that are not satisfied by the current γ. This step also provides a solution to $\mathcal{I}$, which may improve on the current upper bound UB. This is why the termination criterion is checked on Line 13 before the next iteration.

The `ExtractCores` subroutine is detailed as Algorithm 2. Given a hitting set γ over $\mathcal{C}$, the procedure iteratively invokes a PB solver on $\mathcal{I}$ under a set $\gamma_a \subset \gamma$ of assumptions (viewed here as a set of literals) via the procedure `PB-SolveUnderAssumptions` on Line 5. This PB solver call provides either a set $\mathcal{K}$ of core constraints of $\mathcal{I}$ that are not satisfied by γ, or a solution τ to $\mathcal{I}$ for which $\gamma_a \subset \tau$. In the first case, the set of assumptions is refined and the loop reiterated. In the latter case, the cost of the solution $\mathrm{O}(\tau)$ is compared to the current UB which is updated if needed. For refining the set of assumptions between solver calls, the procedure makes use of weight-aware core extraction (WCE) originally proposed in the context of core-guided MaxSAT solving [3]. WCE generalizes the so-called disjoint cores technique—iteratively computing a set of variable-disjoint core constraints—by taking into account the weights of literals. Initially, a temporary weight $\mathcal{W}(x)$ of each variable $x \in \mathrm{VAR}(\mathrm{O})$ is initialized according to the coefficient of the corresponding literal in O. After extracting a core constraint C, the weight of each variable $x \in \mathrm{VAR}(C)$ is lowered by the minimum weight of each variable appearing in C. In the next iteration, the set of assumptions is refined to contain the literals of non-negative temporary weights. Notice that on each invocation of Line 13, the temporary weight of at least one variable is lowered to 0 or less. As such, the size of the set γ_a of assumptions is decreased in each iteration, which implies the termination of `ExtractCores`.

4 Improvements to PBO-IHS

With details on the PBO-IHS approach in place, in this section we detail further refinements to the `PBO-IHS` algorithm. In particular, we propose the integration of solution-improving search into the PBO-IHS main search loop, with the potential of obtaining further improved upper bounds and thereby speeding up search. Furthermore, we consider alternative strategies for extracting core constraints with a PB solver, including making use of recent developments which allow for obtaining more general core constraints than the clausal (at-least-one cardinality) constraints employed originally in PBO-IHS.

```
1  PBO-IHS+SIS(𝓘)
       Input: A PBO instance 𝓘
       Output: An optimal solution τ
2      (τ_best, sat?) ← PB-Solve(𝓘);
3      if not sat? then return "no feasible solutions";
4      UB ← O(τ_best); LB ← 0;
5      𝒞 ← SeedConstraints(𝓘, 𝒞);
6      while TRUE do
7          (γ, opt?) ← ComputeHittingSet(O, UB, 𝒞);
8          if opt? then LB ← O(γ);
9          if UB = LB then break;
10         if Schedule-SIS() then SIS(UB, τ_best, O, sis-bound);
11         if UB = LB then break;
12         𝒞 ← 𝒞 ∪ ReducedCostFixing(O, UB, 𝒞);
13         𝒞 ← 𝒞 ∪ ExtractCores(γ, UB, τ_best, 𝓘);
14         if UB = LB then break;
15     return τ_best
```

Algorithm 3 The PBO-IHS algorithm with integrated solution-improving search.

4.1 Integrating Solution-Improving Search into PBO-IHS

We start with the integration of solution-improving search into PBO-IHS. Solution-improving search (SIS; also known as model-improving search) has shown to be an effective approach to MaxSAT solving [28, 2], along with the core-guided and IHS approaches and has also been applied in PBO [4, 32, 13]. Here we propose to integrate SIS into PBO-IHS, resulting in essentially what can be considered an IHS-SIS hybrid for PBO.

Given a PBO instance $\mathcal{I} = (F, \mathrm{O})$ and a current upper bound UB on $\mathrm{O}(\mathcal{I})$, solution-improving search invokes a PB solver on the instance $F \cup \{\sum_{(w,l)\in \mathrm{O}} w \cdot l < UB\}$, querying for a solution to $\mathcal{I}$ with better cost than UB. If the result is satisfiable, the solver provides a solution τ for which $\mathrm{O}(\tau) < UB$. Thereby the current upper bound UB is improved. If the result is unsatisfiable, $\mathrm{O}(\mathcal{I}, \tau) \geq UB$ holds for all solutions τ of $\mathcal{I}$, proving that $\mathrm{O}(\mathcal{I}) = UB$. While SIS is a complete algorithm for PBO in the sense that it will eventually compute an optimal solution if a PB solver is invoked iteratively under an increasingy strict UB, here a main motivation for integrating SIS into PBO-IHS is that each iteration of SIS either finds an optimal solution to $\mathcal{I}$, or lowers the known upper bound on $\mathrm{O}(\mathcal{I})$. The improved bound can then in turn be exploited by the other parts of the `PBO-IHS` algorithm.

Algorithm 3 details `PBO-IHS+SIS`, the `PBO-IHS` algorithm with integrated solution-improving search. The necessary changes to the base algorithm are highlighted in blue. After computing a hitting set and potentially updating the current lower bound, Algorithm 3 makes a heuristic decision using `Schedule-SIS` on whether to enter into a solution-improving search phase at this stage (Line 10). When entering the solution-improving search phase, SIS is employed until either an optimal solution for $\mathcal{I}$ is found or a resource-limit `sis-bound` allocated to SIS at this iteration is exceeded. (We study the impact of different resource limits and heuristic choices for `Schedule-SIS` in the empirical part of this work.) The procedure `SIS` updates the upper bound and the best known model τ_{best} if it finds an improved solution. Underlining the benefits of SIS, note that even if a SIS invocation is not able to *prove* the optimality of a solution computed (i.e., even though SIS would find an optimal solution, the next unsatisfiable PB solver call querying for an even better solution, could be terminated

early due to the resource-limit), it may still be able to provide a solution that turns out to be optimal by simply afterwards checking that the cost of the best solution obtained from SIS matches the current lower bound maintained by PBO-IHS that is obtained though hitting set computation. This will allow PBO-IHS integrated with SIS to terminate on Line 11.

4.2 Extracting Core Constraints

In addition to the integration of SIS into PBO-IHS, we explore potential improvements to core extraction in PBO-IHS. After reviewing the `PB-SolveUnderAssumptions` algorithm used in the original version of `PBO-IHS` [35], we consider an alternative, more general strategy for extracting core constraints with a conflict-driven PB solver supporting assumptions [18], which allows for extracting more general core constraints. Furthermore, we consider a lightweight approach to extracting multiple cores via a single PB solver call using what we refer to as assumption shuffling.

4.2.1 Conflict-Driven PB Solving

For understanding the similarities and differences between the core extraction approaches we consider, we start with an overview of the so-called conflict-driven constraint learning paradigm for pseudo-Boolean solving. For a more detailed description of conflict-driven PB search, we refer the reader to [18].

Given a formula F and a set γ of assumptions (viewed as a set of literals) the goal of a conflict-driven PB-solver is to compute an extension $\tau \supset \gamma$ that is a solution to F or—more central to our discussion—to prove that no such extension exists. During search, a conflict-driven PB solver maintains a working formula $F^w \supset F$ (initialized to F), and an ordered sequence (a *trail*) ρ of literals such that the first $|\gamma|$ literals of ρ are the literals in γ in some order. We say that a literal l is assigned to 1 by ρ if $l \in \rho$ and to 0 if $\bar{l} \in \rho$, and otherwise that l is unassigned by ρ. The PB solver attempts to extend ρ into a solution to F by alternating between decision and propagation steps, much alike CDCL SAT solvers. A propagation step consists of extending ρ with a literal l for which there is a constraint $C \in F^w$ that, informally speaking, would be falsified by ρ if it was extended with $\bar{l}$ instead. We say such l is propagated and its reason $\texttt{reason}(l)$ is the constraint C. If no literals can be propagated, the trail is instead extended by heuristically selecting a literal l that is unassigned by ρ. Such l is called a decision literal.

The decision and propagation steps continue until the current trail ρ either (a) constitutes a solution to $F^w \supset F$ or (b) falsifies some constraint $D \in F^w$. In case (a) ρ is a solution to F under the assumptions γ. In case (b) the algorithm next performs conflict analysis to learn a new constraint C' that is falsified by the current trail, but satisfied by any solutions of F. The new constraint is added to F^w before backjumping i.e., removing enough literals from the trail for C' not to be falsified anymore. If conflict analysis learns a constraint C^{CONFL} and the search backjumps to a trail containing only assumptions and propagated literals which still falsify the constraint, the search terminates since then there are no solutions of F that extend γ. At this point core constraints of the PB-instance at hand can be extracted by performing conflict analysis on C^{CONFL}. The following subsections detail different strategies for doing so.

We note that this description of the conflict driven paradigm in PB is somewhat simplified. Actual implementations of such solvers make use of the fact that a PB constraint can be falsified already before all literals in it are assigned. This requires some non-trivial alterations to conflict analysis in order to guarantee that invariants required for correctness hold.

4.2.2 Clausal Cores

Consider a call to `PB-SolveUnderAssumptions` made during Algorithm 2 under which the PB solver is invoked on the constraints F of a PBO instance $\mathcal{I} = (F, \mathrm{O})$ under a set $\gamma \subset \textsc{lit}(\mathrm{O})$ of assumptions. If γ cannot be extended to a solution to F, the solver learns a conflict PB constraint $C^{\textsc{confl}}$ that is falsified by the assumptions γ and any literals propagated by them. The method for extracting core constraints employed in [35] proceeds by—starting from $\kappa = \textsc{lit}(C^{\textsc{confl}})$—iteratively exchanging each propagated literal $l \in \kappa$ by the negations of the literals in the reason for $\bar{l}$, i.e., $\textsc{lit}(\texttt{reason}(\bar{l})) \setminus \{\bar{l}\}$. As $C^{\textsc{confl}}$ is falsified, the negation of each literal in $C^{\textsc{confl}}$ is either an assumption or a propagated literal. The procedure stops when only assumption literals remain. At that point $\textsc{var}(\kappa) \subset \textsc{var}(\gamma) \subset \textsc{var}(\mathrm{O})$ and the constraint $\sum_{l \in \kappa} l \geq 1$ is satisfied by all solutions of $\mathcal{I}$. Thus it is a core constraint of $\mathcal{I}$. The core $\sum_{l \in \kappa} l \geq 1$ extract is always an at-least-one constraints and hence, similar to cores in SAT, can be expressed as a single CNF clause. We refer to this approach to core constraint extraction as `Clausal-Cores`.

4.2.3 More General PB Cores

The second—and more general—method of extracting cores works by directly reasoning on the PB constraints, and is earlier implemented in a core-guided PBO solver [14]. However, it should be noted that in the context of core-guided search, the more general PB core constraints obtained are still rounded to obtain an at-least-k cardinality constraint for some $k \in \mathbb{N}$. In contrast, here we will employ the more general cores as generalized hitting set constraints in the context of PBO-IHS.

Starting from $C = C^{\textsc{confl}}$, the approach applies the so-called generalized resolution rule [21, 15] in the realm on PB on each propagated literal $l \in \textsc{lit}(C)$ and its reason $\texttt{reason}(\bar{l})$ in order to obtain a new constraint D that is satisfied by any assignment that satisfies both C and $\texttt{reason}(\bar{l})$ and does not contain the literal l. The process then sets $C = D$ and continues until C does not contain propagated literals. At this point, $\textsc{var}(C) \subset \textsc{var}(\gamma) \subset \textsc{var}(\mathrm{O})$ and C is satisfied by all solutions of $\mathcal{I}$. Thus C is a core constraint of the PBO instance at hand. As this approach to extracting core constraints can result in general PB constraints, we refer to it as `PB-Cores`.

▶ **Example 3.** Consider the PBO instance $\mathcal{I}$ from Example 1 and let $\gamma = \{\overline{a_1}, \overline{a_2}\}$. Assume that the PB solver is invoked on $\mathcal{I}$ under the assumptions γ. The assumptions propagate $\overline{x_1}$ and $\overline{x_2}$ and (e.g.) x_3. The trail $\langle \overline{a_1}, \overline{a_2}, \overline{x_1}, \overline{x_2}, x_3 \rangle$ falsifies the constraint $x_2 + \overline{x_3} \geq 2$ so the solver enters conflict analysis and learns for example the constraint $C = x_1 + x_2 \geq 2$. After backtracking, the assumptions $\{\overline{a_1}, \overline{a_2}\}$ propagate $\overline{x_1}$ with the reason $\texttt{reason}(\overline{x_1}) = a_1 + \overline{x_1} \geq 1$ and $\overline{x_2}$ with the reason $\texttt{reason}(\overline{x_2}) = a_2 + \overline{x_2} \geq 1$. The trail $\langle \overline{a_1}, \overline{a_2}, \overline{x_1}, \overline{x_2} \rangle$ falsifies C so the search terminates and returns UNSAT.

The `Clausal-Cores` core extraction method now initializes $\kappa = \textsc{lit}(C) = \{x_1, x_2\}$. It then iteratively processes both literals in κ, first replacing x_1 by a_1 and then x_2 by a_2. At this point $\kappa = \{a_1, a_2\}$ contains only assumptions, so the core constraint $a_1 + a_2 \geq 1$ is learned.

In contrast, the `PB-Cores` core extraction method initially resolves C with $\texttt{reason}(\overline{x_1})$ on x_1 to obtain $C' = a_1 + x_2 \geq 2$. Then it resolves C' with $\texttt{reason}(\overline{x_2})$ on x_2 in order to obtain the stronger core constraint $a_1 + a_2 \geq 2$.

Note that `PB-Cores` core extraction method is indeed a generalization of `Clausal-Cores`: `Clausal-Cores` can be viewed as a special case of `PB-Cores` that treats every PB constraint

in the inference (resolution) steps as a clause, an idea which has been explored for general PB solving already in [4].

4.2.4 More Cores via Assumption Shuffling

Assumption shuffling allows for extracting multiple core constraints after the final conflict analysis performed by `Clausal-Cores` and `PB-Cores`. The intuition underlying assumption shuffling is that the final constraint C^{CONFL} learned by the PB solver might be falsified by (the literals propagated by) several different subsets of the current assumptions γ. The specific core constraint obtained during conflict analysis depends then on which constraints are marked as reasons for the propagated literals. This in turn depends on the order in which the assumption literals are added to the trail. As such, an inexpensive way of attempting to learn more than one core constraint is to permute the order in which assumptions are added to the trail and reperforming the final conflict analysis step (either with `PB-Cores` or `Clausal-Cores`). Since the PB solver has already learned a constraint C^{CONFL} that is falsified by the assumptions regardless of which order they are added to the trail, each permutation and core constraint extraction can be done in polynomial time. Essentially this is a lightweight way of extracting potentially multiple core constraints per PB-solver call within PBO-IHS, and thereby also potentially tightening the hitting set instance more quickly. We call this technique *assumption shuffling*.

▶ **Example 4.** Assume that the PB solver is invoked under the assumptions $\gamma = \{\overline{a_1}, \overline{a_2}, \overline{a_3}\}$ and the solver learns the constraints $\{a_1{+}x \geq 1, a_2{+}x \geq 1, a_3{+}y \geq 1\}$ and $C^{\text{CONFL}} \equiv \overline{x}{+}\overline{y} \geq 1$. The constraint is falsified both by $\gamma_1 = \{\overline{a_1}, \overline{a_3}\}$, and $\gamma_2 = \{\overline{a_2}, \overline{a_3}\}$ as both propagate the literals $\{x, y\}$. Propagating γ_1 (i.e. $\overline{a_1}$ before $\overline{a_2}$) obtains the core constraint $a_1 + a_3 \geq 1$ regardless of the core extraction method used. In contrast, propagating γ_2 (a_2 before a_1) obtains the core constraint $a_2 + a_3 \geq 1$.

5 Empirical Evaluation

To evaluate the ideas and techniques described in the previous section, we implemented them into the original version of PBO-IHS. In this section we report on an extensive empirical evaluation of the impact of these techniques on the efficiency of PBO-IHS.

The experiments were run on computing nodes with 8-core Intel Xeon E5-2670 2.6-GHz CPUs and 64-GB RAM under a per-instance 3600-second time and 16-GB memory limit.

5.1 Implementation

The PBO-IHS solver is implemented in Python, using Roundingsat version 2 [18] (commit 1476bf0bcd) as the PB solver and IBM ILOG CPLEX version 12.8 via its C++ API for hitting set computation. The open-source implementation and empirical data are available available at `https://bitbucket.org/coreo-group/pbo-ihs-solver/`. We implemented solution-improving search into PBO-IHS using a separate instantiation of the PB solver RoundingSAT. This is due to the fact that, as we observed in preliminary experimentation, the upper-bounding constraints enforced for solution-improving search can cause unnecessary runtime overheads in core extraction if the same PB solver instantiation would be used for both the core extraction and SIS steps.

The `Clausal-Cores` strategy for extracting cores, employed originally in PBO-IHS, is implemented by extending Roundingsat to include an analyzeFinal function similar to the one implemented in MiniSat-like SAT solvers [16, 17]. We implemented the `PB-Cores` core

extraction strategy into PBO-IHS by using the core extraction implementation from [14], with the modification that we disabled the final rounding step applied in [14] to obtain a cardinality constraint (necessary for employing the core in the core-guided PBO setting [14]).

In the evaluation, we consider the following variants of PBO-IHS.

- `PBO-IHS:CLAUSAL`: the original PBO-IHS implementation, as presented in [35] (using the `Clausal-Cores` core extraction strategy).
- `PBO-IHS:CLAUSAL+SIS`: `PBO-IHS:CLAUSAL` with integrated solution-improving search.
- `PBO-IHS:PB`: `PBO-IHS:CLAUSAL` using `PB-Cores` instead of `Clausal-Cores` as the core constraint extraction strategy.
- `PBO-IHS:PB+SIS`: `PBO-IHS:PB` with integrated solution-improving search.
- `PBO-IHS:CLAUSAL&PB`: `PBO-IHS:CLAUSAL` using both `Clausal-Cores` and `PB-Cores` core constraint extraction step strategies (with separate WCE steps for each strategies).
- `PBO-IHS:CLAUSAL&PB+SIS`: `PBO-IHS:CLAUSAL&PB` with integrated solution-improving search.

All variants employ assumption shuffling for extracting several cores at each iteration. The default parameters used are the following: assumptions are shuffled 5 times (to obtain 5 cores); the resource-limit parameter `resource-bound` for SIS per main search loop iteration was enforced as the time limit of at most 30 seconds; and as an implementation of the `Schedule-SIS` heuristic for triggering SIS (recall Algorithm 3), we trigger SIS whenever the lower bound LB has not improved during the last 5 main search loop iterations. We will separately provide empirical evidence on the impact of the parameter, showing that these default choices appear to be suitable for good overall performance.

5.2 Competing Approaches

A comprehensive empirical evaluation of the original version of PBO-IHS was presented in [35] using the same empirical setup and computing hardware as used for the experiments reported here. The earlier empirical evaluations showed that PBO-IHS outperforms other competing specialized approaches to PBO (including Open-WBO [25], Sat4J [4], NaPS [32], Roundingsat (RS) [18], RS/lp [13] and RS/oll, also known as HYBRID [14]) and also that PBO-IHS provides complementary performance when compared with CPLEX [8]. We refer the reader to [35] for more details.

Since the closest competitor among the specialized PBO solvers in terms of runtime performance was shown to be **RS/lp** [13], we also report on its performance for reference here. **RS/lp** is an implementation of (pure) solution-improving search that periodically invokes a linear programming (LP) solver on the LP relaxation of the instance being solved. The LP calls are used for deriving more conflicts withing the conflict-driven procedure implemented in 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.)

5.3 Benchmarks

We use the same set of benchmarks as in the work reporting on the original version of PBO-IHS [35]. In particular, we use all benchmarks from Pseudo-Boolean Competition 2016 [29] (as the most recent instantiation of the competition) and all 0-1 integer programs from the MIPLIB 2017 library [20] and earlier MIPLIB releases, filtering out all unsatisfiable benchmarks, benchmarks without objective functions and benchmarks with very large ($\geq 2^{64}$) coefficients. The filtered set contains 8456 benchmarks from the Pseudo-Boolean competition

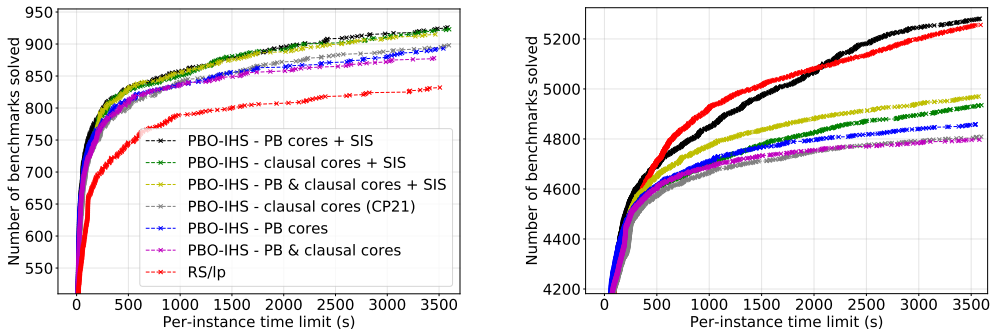

Figure 2 Effect of solution improving search and different core extraction strategies on the sampled (left) and full (right) benchmark sets.

and 252 benchmarks from MIPLIB, respectively. A breakdown of the full benchmark set into domains is detailed as part of Table 1. Furthermore, following the approach of used in [35], due to the fact that the number of benchmarks per domain included in the benchmark set varies significantly depending on the domain, 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 experiments. The sampled benchmark set contains in total 1786 benchmarks. Unless explicitly stated otherwise, the results reported on in this section are with respect to the sampled benchmark set. This allows us to present e.g. so-called runtime distributions over all benchmarks without fear of any single benchmark domain dominating the results and conclusions drawn. We note that, as shown in [35], the empirical results can be expected to be robust with respect to different random samplings.

5.4 Results: Impact of SIS and Core Extraction Strategies

We first consider the impact of SIS and different core extraction strategies on the runtime performance of PBO-IHS, using default parameters. Figure 2 (left: sampled benchmark set, right: full set) shows the number of benchmarks solved (y-axis) under different per-instance time limits (x-axis) for the different variants of `PBO-IHS`. Focusing on the sampled benchmark set, we observe that the integration of SIS has a clear performance boosting effect on PBO-IHS. This holds regardless of the choice of core extraction strategy. `PBO-IHS:PB+SIS` solves the greatest number of instances among the variations. Compared to the integration of SIS, the different core extraction strategies result in—perhaps even surprisingly—similar performance, suggesting that the more general core constraints do not appear to be significantly beneficial for overall performance when considering a balanced sample of benchmarks from all domains.

There are, however, benchmark domains on which the core extraction strategies have a significant impact. In particular, as shown in Figure 2 right for the full set of benchmarks, dominated by few domains of significant size, `PBO-IHS:PB+SIS` using the `PB-Cores` strategy and SIS significantly outperforms the other PBO-IHS variants (and also surpassing **RS/lp**—which PBO-IHS regardless of the variant outperforms on the balanced set of benchmarks). Table 1 provides a more detailed breakdown of these results per benchmark domain and offers an explanation for the good performance of `PBO-IHS:PB+SIS`: the `PB-Cores` core extraction strategy and SIS together significantly improve performance in particular on the BA and NG domains, two of the three largest domains in terms of number of benchmarks. As employing `PB-Cores` does not result in noticeably worse performance that `Clausal-Cores` on

Table 1 Comparison of specialized PBO solver per benchmark domain: number of solved instances (#) and cumulative runtimes over solved instances in seconds (cum.)

	clausal cores (CP21)		clausal cores + SIS		PB cores		PB cores + SIS		PB & clausal cores		RS/lp	
Domain (#instances)	#	cum.	#	cum.	#	cum.	#	cum.	#	cum.	#	cum.
10orplus/9orless (156)	156	23670	156	**6125**	156	6378	156	6275	156	18616	154	55344
caixa (24)	24	64	24	**32**	24	33	24	33	24	42	24	70
rand.*list (118)	118	2296	118	1200	118	1222	118	1220	118	1803	118	**692**
area_* (59)	51	11784	50	6513	55	8661	55	11487	**56**	7874	54	16176
trarea_ac (18)	18	7751	18	**2508**	18	2524	18	2701	18	2550	16	3722
aries-da_nrp (70)	32	10413	34	19124	31	7600	33	8244	31	5712	**43**	15442
BA (1440)	20	30038	124	159568	78	79760	406	533565	52	72832	**588**	472938
NG (960)	0	0	0	0	8	18261	**74**	130170	1	1791	48	115499
MANETs (150)	25	21152	26	31446	20	10110	21	13585	25	21192	**40**	23547
BioRepair (30)	30	262	30	222	30	**207**	30	213	30	240	30	3258
Metro (30)	27	12595	29	11116	22	21867	28	2078	23	14373	**30**	1795
ShiftDesign (30)	9	9060	9	1850	17	15996	17	16370	9	9751	**18**	12824
Timetabling (30)	28	8768	27	6960	26	2820	27	10265	28	**5981**	23	15419
EmployeeScheduling (14)	0	0	0	0	0	0	0	0	0	0	0	0
golomb-rulers (34)	12	4212	12	4965	12	5149	12	5010	12	4608	12	**1216**
bsg (60)	5	16	10	607	6	1470	10	838	7	709	10	**465**
mis/mds (120)	**57**	15335	55	10350	54	5223	54	7646	54	8732	45	3853
course-ass (6)	1	6	3	29	2	9	3	**14**	1	1	3	33
decomp (10)	0	0	0	0	0	0	0	0	0	0	0	0
data (68)	11	2163	14	1423	11	1136	**16**	1133	12	2884	13	4044
dt-problems (60)	60	113	60	34	60	35	60	34	60	70	60	**2**
domset (15)	0	0	0	0	0	0	0	0	0	0	0	0
factor (186)	186	342	186	166	186	122	186	118	186	202	186	**2**
factor-mod-B (225)	225	344	225	173	225	166	225	199	225	251	225	**60**
fctp (35)	12	**499**	12	651	12	648	12	646	12	649	5	940
featureSubscription (20)	20	303	20	329	20	289	20	792	20	**239**	20	8106
frbXX-XX-opb (40)	6	11343	6	9032	6	9025	6	9024	6	**8986**	0	0
flexray (9)	4	**50**	4	90	4	87	4	88	4	90	4	393
fome (3)	0	0	0	0	0	0	0	0	0	0	0	0
graca (100)	84	40593	77	49457	76	21482	72	55416	84	**26730**	31	21769
haplotype (8)	7	4023	8	**2212**	6	1486	8	5529	6	602	8	2385
garden (7)	6	76	6	48	6	**47**	6	48	6	50	5	1
hw32/hw64/hw128 (27)	**10**	12063	9	10233	7	3134	7	3102	7	3210	8	3470
jXXopt (2040)	1579	64191	1579	75181	1575	59765	1575	58216	1574	55900	**1589**	51821
keeloq_tasca (4)	4	54	4	57	4	78	4	**26**	4	63	4	33
kullmann (7)	3	3016	3	2942	3	2933	3	2926	3	**2909**	1	2
lion9-single-obj (1513)	1487	120526	**1489**	133582	1487	108929	1488	134095	1486	89779	1412	113829
logic-synthesis (74)	71	708	71	665	71	**650**	71	650	71	669	61	11647
miplib/neos (79)	38	10631	40	10323	39	9868	41	13818	41	**10596**	37	8377
miplib/other (405)	156	38501	**166**	88369	156	51454	163	59089	159	45063	147	36264
unibo (36)	8	5342	10	9219	9	5596	10	**8154**	9	5573	3	228
market-split (20)	1	1167	1	3600	0	0	0	0	0	0	**4**	342
opb/graphpart (31)	24	5211	**25**	8550	24	4657	24	6924	24	5121	12	435
opb/autocorr_bern (43)	8	2089	9	5053	9	5048	9	5056	9	**5047**	4	3594
opb/sporttournament (22)	11	3121	11	1240	11	**1238**	11	1597	11	1307	4	23
opb/edgecross (19)	12	3984	12	2591	12	**2581**	12	2809	12	4333	6	2899
opb/pb (8)	0	0	0	0	0	0	0	0	0	0	0	0
opb/faclay (10)	1	**960**	1	3278	1	3273	1	3268	1	3248	0	0
opb/other (6)	1	2	1	0	1	1	1	1	1	1	1	**0**
primes/aim (48)	46	234	46	194	46	188	46	189	46	200	**48**	4
primes/jnh (16)	16	53	16	40	16	37	16	37	16	42	16	**19**
primes/ii (41)	**34**	5230	33	1582	33	1657	33	1573	33	1577	23	6874
primes/par (30)	20	422	20	266	20	265	20	265	20	269	20	**15**
primes/other (13)	5	938	5	1089	5	1088	5	1085	5	1078	**6**	452
routing (15)	15	26	15	12	15	12	15	12	15	18	15	**7**
radar (12)	12	77	12	85	12	64	12	**63**	12	65	6	71
synthesis-ptl-cmos (10)	10	16	10	7	10	7	10	**7**	10	9	9	135
testset (6)	6	8	6	4	6	4	6	3	6	5	6	**0**
ttp (8)	2	10	2	6	2	20	2	3	2	15	2	**0**
vtxcov (15)	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
Total: 8708	4814	495850	4939	684400	4863	484356	**5286**	1125708	4843	453657	5257	1020531

Table 2 Effect of different SIS-related parameters on the number of solved instances.

		Number of solved instances			
		# seeds			full benchmark set
Parameter	**value**	≥ 5	≥ 1	≥ 10	
no SIS (baseline)		892	903	888	4860
`resource-bound` (s)	10	923	930	918	5242
	30	**929**	**936**	**925**	**5282**
	60	923	928	913	5276
	90	923	927	915	5274
	180	919	923	909	5247
`#iter`	**5**	**929**	**936**	**925**	**5282**
	10	926	933	921	5240
	25	922	929	916	5167
	50	916	925	913	5081
`bound`	***LB***	**929**	**936**	**925**	**5282**
	UB	927	935	923	5274
	LB&*UB*	924	932	920	5285

any problem domain, `PB-Cores` is the preferred core constraint extraction strategy (note also that the combination of the two strategies appears to consistently lead to weaker performance then either of the individual strategies—likely due to the overhead involved in applying both).

We also observe synergetic benefits of employing both `PB-Cores` and SIS on specific domains: employing SIS together with `PB-Cores` increases the number of benchmarks solved in the NG domain by 66, while employing SIS together with `Clausal-Cores` does not increase the number of solved instances. In the BA domain, SIS increases the number of solved benchmarks by 328 when using `PB-Cores`, compared to 104 when using `Clausal-Cores`.

5.5 Results: Impact of SIS Parameters

Next, we consider the impact of the parameters involved in scheduling SIS: how often to invoke solution improving search (`Schedule-SIS` of Algorithm 3), and the time limit enforced on SIS per iteration (the `resource-bound` parameter of Algorithm 3).

For an in-depth investigation, we used 10 different random samplings of the full benchmark set (as described in Section 5.3). We report the number of benchmarks solved in at least 1 (virtual best performance), 5 (median number of solved instances), and 10 (i.e., all) of the sampled benchmark sets.

For the time limit enforced on SIS invocations, we considered 10, 30, 60, 90, 180 seconds, keeping the default parameter for when to schedule SIS. For scheduling SIS, recall that the default SIS trigger is that the search stagnated in terms of non-improving lower bound *LB* for 5 main search loop iterations. Here we consider two alternative stagnation criteria: (i) *UB* (instead of *LB*) does not improve, and (ii) neither *UB* nor *LB* improves for 5 iterations. We also consider varying the number of iterations after which to trigger SIS if bounds-information is stagnated, considering as the iteration-threshold the values 10, 25, and 50 in addition to the default value 5. For the experiments, we varied individually the time limit, stagnation criterion and iteration-threshold, keeping the other parameters in the default values.

The results are shown in Table 2. For the `resource-bound` parameter, we observe that

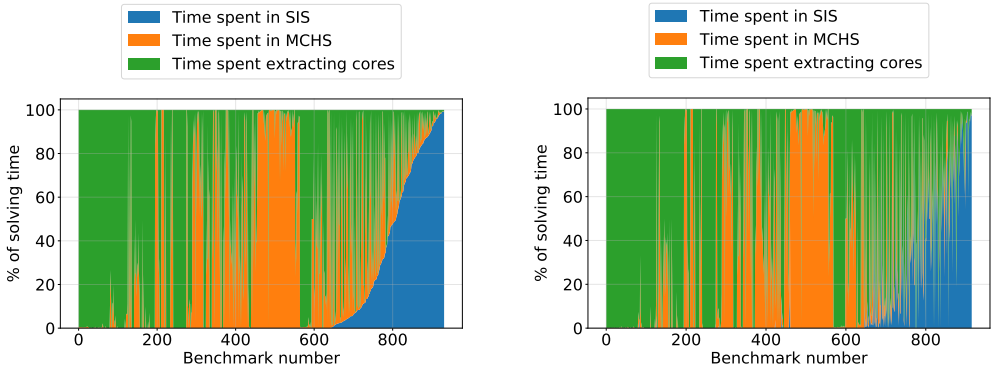

Figure 3 Runtime division between solution improving search (SIS), hitting set computation (MCHS) and core extraction for `PBO-IHS:PB+SIS` (left) and `PBO-IHS:CLAUSAL+SIS` (right).

enforcing a 30-second bound on the individual SIS calls results in the largest number of instances solved. The number of solved benchmark decreases when the time limit is increased. This suggests that it is indeed important to limit the runtime resources allocated to SIS. For the number of iterations allowed before triggeting SIS at stagnation (`#iter` in the table), we clearly observe that detecting stagnation early and employing SIS for improving the current upper bound is beneficial for overall performance. Finally, for the stagnation criterion (`bound` in the table), we observe that best-performance is indeed achieved when solely using LB for detecting stagnation; this is inline with the intuition that the base PBO-IHS algorithm (as a pure IHS approach) is mainly a lower-bound based algorithm, which SIS can improve on by providing tighter upper bound information.

5.6 Results: Runtime Division among Core Extraction, MCHS and SIS

Next we consider how the overall runtimes of PBO-IHS with integrated SIS divide among the three main phases of PBO-IHS, namely, core extraction, hitting set computation (MCHS), and SIS. Figure 3 shows a distribution of the percentage of runtimes spent in the three phases for `PBO-IHS:PB+SIS` (left) and `PBO-IHS:CLAUSAL+SIS` (right) over the sampled benchmark set. The benchmarks are sorted by increasing fraction of the total time spent in SIS by `PBO-IHS:PB+SIS`, i.e., when using the `PB-Cores` core extraction strategy.

First, we observe the two plots are very similar, suggesting that the choice of the core extraction strategy does not influence the runtime distributions significantly. That said, using `Clausal-Cores` appears to somewhat increase the percentage of time spent on core extraction on instances on which `PBO-IHS:PB+SIS` spends a noticeable fraction of the runtime in SIS. Second, we observe that time spent in SIS is significant for only a minority of the benchmarks, suggesting that the base PBO-IHS algorithm does not stagnate significantly on a majority of the instances. However, the earlier discussed runtime comparison suggests that SIS does have a significant positive impact on instances it consumes a greater percentage of the overall runtime. Thirdly, we observe that the percentage of runtime spent in hitting set computation varies significantly between different instances.

5.7 Results: Impact of Assumption Shuffling

Finally, we evaluate the impact of assumption shuffling as a lightweight way of obtaining several cores on the runtime performance of `PBO-IHS:PB+SIS`. For this experiment, we varied

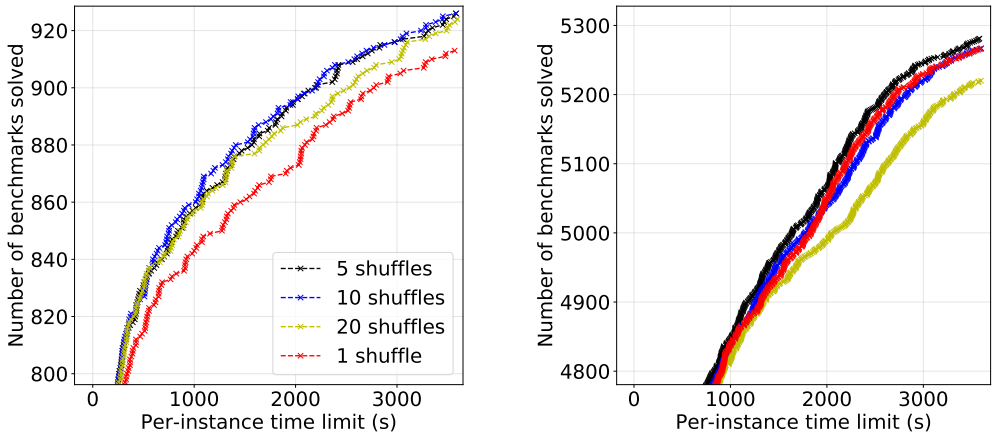

Figure 4 The effect of varying the number of assumption shuffles in each core extraction call over the sampled (left) and full (right) benchmark set.

the number of shuffles between 1 (no shuffling), 5 (default), 10, and 20. As seen from Figure 4 assumption shuffling improves on the number of solved instances, regardless of the choice between the number of shuffles (5, 10, or 20). The number of shuffles appears less significant in terms of the number of solved instances. However, it appears that a smaller number of shuffles may be enough and even better than a greater number; we suspect that this has to do with the fact that, as the number of shuffles is increased, the size of the minimum-cost hitting set instance becomes larger faster. This effect is especially noticeable over the full benchmark set (shown in the right side Figure 4) where 20 shuffles leads to noticeably fewer solved instances.

6 Conclusions

A first instantiation PBO-IHS of the implicit hitting set approach in the context of pseudo-Boolean optimization was recently developed and shown to outperform earlier specialized PBO solvers. In this work we studied ways of further improving the overall runtime performance of PBO-IHS. In particular, we propose the integration of solution-improving search (SIS) to PBO-IHS, essentially obtaining a hybrid IHS-SIS approach to PBO, and showed that this hybrid pushes the empirical performance of PBO-IHS significantly further. We also studied the impact of different core constraint extraction strategies as a central part of IHS search, including the employments of recent advances in PB core extraction, which allows for making use of more general core constraints, resulting in solving a generalized form of the hitting set problem within PBO-IHS. In addition to SIS, the refined core extraction strategies proved to provide performance-improvements in particular benchmark domains. We also extensively evaluated the runtime impact of different parameters involved in the integration of SIS and core extraction. All in all, the new version of PBO-IHS resulting from this work can be considered a competitive exact solver for PBO.

References

1 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.

2 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.

3 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.

4 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.

5 Sam Buss and Jakob Nordström. *Proof complexity and SAT solving*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 7, pages 133–182. IOS Press BV, 2021. doi:10.3233/FAIA200990.

6 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.

7 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.

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

9 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.

10 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.

11 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.

12 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.

13 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.

14 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.

15 Heidi E. Dixon, Matthew L. Ginsberg, and Andrew J. Parkes. Generalizing boolean satisfiability I: background and survey of existing work. *J. Artif. Intell. Res.*, 21:193–243, 2004. doi:10.1613/jair.1353.

16 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.

17 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.

18 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.

19 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.

20 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.

21 John N. Hooker. Generalized resolution for 0-1 linear inequalities. *Ann. Math. Artif. Intell.*, 6(1-3):271–286, 1992. doi:10.1007/BF01531033.

22 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.

23 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.

24 Mikolás Janota, António Morgado, José Fragoso Santos, and Vasco M. Manquinho. The seesaw algorithm: Function optimization using implicit hitting sets. In *CP*, volume 210 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

25 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.

26 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001. doi:10.1145/378239.379017.

27 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.

28 Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic polynomial watchdog encoding for solving weighted maxsat. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference,*

SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings, volume 10929 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2018. doi:10.1007/978-3-319-94144-8_3.

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

30 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.

31 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.

32 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.

33 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.

34 João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999. doi:10.1109/12.769433.

35 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-boolean optimization by implicit hitting sets. In *CP*, volume 210 of *LIPIcs*, pages 51:1–51:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.