Oracle-Based Local Search for Pseudo-Boolean Optimization

Markus Iser[a], Jeremias Berg[a] and Matti Järvisalo[a;*]

[a]University of Helsinki, Finland
ORCiD ID: Markus Iser https://orcid.org/0000-0003-2904-232X,
Jeremias Berg https://orcid.org/0000-0001-7660-8061, Matti Järvisalo https://orcid.org/0000-0003-2572-063X

Abstract. Significant advances have been recently made in the development of increasingly effective in-exact (or incomplete) search algorithms—particularly geared towards finding good though not provably optimal solutions fast—for the constraint optimization paradigm of maximum satisfiability (MaxSAT). One of the most successful recent approaches is a new type of stochastic local search in which a Boolean satisfiability (SAT) solver is used as a decision oracle for moving from a solution to another. In this work, we strive for extending the success of the approach to the more general realm of pseudo-Boolean optimization (PBO), where constraints are expressed as linear inequalities over binary variables. As a basis for the approach, we make use of recent advances in practical approaches to satisfiability checking pseudo-Boolean constraints. We outline various heuristics within the oracle-based approach to anytime PBO solving, and show that the approach compares in practice favorably both to a recently-proposed local search approach for PBO that is in comparison a more traditional instantiation of the stochastic local search paradigm as well as a recent exact PBO approach when used as an anytime solver.

1 INTRODUCTION

Significant progress in developing practical solvers for maximum satisfiability (MaxSAT) [3]—as the optimization extension of the Boolean satisfiability (SAT) problem—has established MaxSAT as a viable choice for a declarative optimization paradigm, enabling efficiently solving various types of real-world combinatorial optimization problems. Leveraging on the extraordinary success of SAT solvers [24, 6, 23, 31, 25] as "real-life NP oracles", together with non-trivial algorithmic advances, research on MaxSAT solving techniques has until recently mostly focused on complete (or exact) algorithms, yielding solvers which are guaranteed to provide provably-optimal solutions given enough computational resources. However, due to intrinsic computational barriers in scaling and speeding up exact approaches in general, the development of practical *incomplete* (or in-exact) MaxSAT solvers has recently gained significant traction [26, 4, 7, 1, 10]. In contrast to typical complete solvers (such as those based on the core-guided, implicit hitting set, model-improving or branch-and-bound approaches [3, 23]), the so-called "incomplete" solvers are in particular geared towards finding relatively good solutions as fast as possible (without general guarantees on optimality), effectively acting as anytime algorithms.

* Corresponding Author. Email: matti.jarvisalo@helsinki.fi.

Beyond pure stochastic local search [7, 21], the most effective "incomplete" solvers today are based on clever combinations of complete solvers, such as the Loandra approach that combines the model-improving and core-guided approaches for incomplete solving [4]. A different approach, first proposed in Mrs. Beaver [26] and subsequently improved and extended in a sequence of publications [28, 27], can be viewed as a non-traditional stochastic local search (SLS) which—somewhat unintuitively yet very successfully—uses a *complete SAT solver* to guide search space traversal from one solution to another, ideally better one. Integrating local search with complete solvers has been shown to increase efficiency in other contexts as well, such as in SAT solvers [8] and finite-domain CP solvers [17].

Despite the noticeably recent success of MaxSAT, the underlying modeling language of propositional logic has its limitations. This holds in particular true for problems in which constraints are more naturally represented as *pseudo-Boolean constraints* [5, 13, 18], i.e., linear inequalities over binary variables, as a central class of integer programs. Complementing the more classical branch-and-cut type approach to solving integer programs, specialized reasoning approaches for pseudo-Boolean constraints have recently been developed [15] building on ideas from MaxSAT solving, extending complete core-guided [12] and implicit hitting set [32, 33] approaches to pseudo-Boolean optimization. However, the analogy between complete and incomplete solving in MaxSAT has received little attention in the realm of PBO, apart from classical stochastic local search approaches [35, 34, 9], with [22] being the most recent development with an open-source solver implementation.

In this work, we investigate the question of whether the success of incomplete MaxSAT solving approaches that use complete SAT solving techniques can be transferred to the PBO domain to obtain a novel type of anytime algorithm for PBO. In particular, we study ways of effectively lifting the Mrs. Beaver approach—implementing non-traditional SAT-oracle based stochastic local search—to the realm of PBO. For this, we make use of the recent RoundingSAT approach [11, 14] as a decision oracle for deciding whether a given set of pseudo-Boolean constraints has a solution under different assumptions. The decision oracle guides the overall search, transitioning from one configuration to another. Stochasticity is introduced by randomization heuristics which affect, e.g., the order in which variables are assigned and the decision oracle is invoked. Furthermore, we integrate a solution-improving search into the approach, which is achieved by enforcing an upper bound constraint—which is naturally declared as a single pseudo-Boolean constraint—in the outer loop of

the procedure. Thereby, the approach can be considered both an anytime approach to PBO and, given sufficient resources, even a complete oracle-based local-search boosted solution-improving approach that, unlike the recently-proposed pure SLS approach to PBO [22], is guaranteed to eventually find an optimal solution. We provide an open-source implementation of the described approach and empirically evaluate its runtime performance. Our implementation of the approach outperforms both a recent pure SLS approach [22] and a recent complete specialized PBO solver running in anytime mode [12] in terms of the quality of solutions found in a short time.

2 PSEUDO-BOOLEAN OPTIMIZATION

A pseudo-Boolean (PB) constraint is of the form $\sum_i c_i x_i \circ B$, where $\circ \in \{\leq, \geq, <, >, \neq, =\}$, each x_i is a binary (0-1) variable, each c_i an integer constant and B an integer bound. Without loss of generality, we assume that every PB constraint is in the normalized form $\sum_i c_i l_i \geq B$, where each coefficient c_i and the bound B are non-negative, and each literal l_i is either a variable x_i or its negation $\bar{x}_i = 1 - x_i$. An assignment M maps binary variables to either 0 or 1. The assignment M satisfies a constraint $\sum_i c_i l_i \geq B$ if $\sum_i c_i M(l_i) \geq B$, where $M(\bar{x}) = 1 - M(x)$. A PB formula F is a set of pseudo-Boolean constraints. An assignment M is a solution to F if it satisfies all constraints in F. When convenient, we view M as the set of literals it maps to 1, i.e., $l \in M$ iff $M(l) = 1$. A solution M is complete for a formula F if it assigns all variables in F, and otherwise partial. An objective $\mathcal{O} \equiv \sum_i c_i x_i$ over binary variables x_i is a pseudo-Boolean expression under minimization. We assume without loss of generality that each coefficient c_i in an objective is positive. We use $\text{VAR}(\mathcal{O})$ to denote the set of variables that appear in an objective $\mathcal{O}$; more precisely, $x \in \text{VAR}(\mathcal{O})$ iff cx is a term in $\mathcal{O}$ for some constant c. A pseudo-Boolean optimization (PBO) instance $\mathcal{I} = (F, \mathcal{O})$ consists of a PB formula F and an objective $\mathcal{O}$. The solutions of $\mathcal{I}$ are the solutions of F. The cost $\mathcal{O}(M)$ of a solution M is the value of $\mathcal{O}$ evaluated under M.

Example 1. Consider the PBO instance $(F, \mathcal{O})$, where $F = \{\sum_{i=1}^{5} b_i \geq 3, b_1 + b_4 \geq 1, b_2 + b_5 \geq 1\}$ and $\mathcal{O} \equiv 3b_1 + 6b_2 + 3b_3 + b_4 + 5b_5$. A minimum-cost solution M of the instance sets $M(b_3) = M(b_4) = M(b_5) = 1$ and $M(b_1) = M(b_2) = 0$ and has $\mathcal{O}(M) = 9$. The solution is represented as a set of literals by $M = \{\bar{b_1}, \bar{b_2}, b_3, b_4, b_5\}$.

3 ANYTIME PBO SOLVING

We consider pseudo-Boolean optimization in an anytime context. In contrast to complete algorithms geared towards computing—given enough time and memory resources—a minimum-cost solution for a given PBO instance, anytime algorithms are essentially designed to find as good solutions as possible in short time.

We discuss two approaches to anytime optimization for PBO relevant to our work: solution-improving search [5, 11] and stochastic local search [20, 22, 35, 34, 9]. The oracle-based local search procedure we develop integrates features from both types of approaches.

3.1 PB Oracles and Solution-Improving PBO Search

Solution-improving search makes iterative queries to a PB decision procedure, abstracted as a PB-oracle `PB-Solve`, in order to iteratively find solutions of increasing quality (i.e., solutions of decreasing cost). A PB-oracle is a complete decision procedure which, given a formula F^w and enough computational resources, either outputs a solution to F^w or determines that one does not exist. Due to recent advances in PB solving, efficient PB solvers implementing a form of a conflict-driven pseudo-Boolean decision procedure [11, 14, 15] are readily available, which is also what we will rely on as the practical PB-oracle in this work. Such an algorithm can be viewed as a lifting of the immensely successful conflict-driven clause learning paradigm from SAT solving into the pseudo-Boolean domain, with native pseudo-Boolean constraint reasoning integrated. In short, a conflict-driven pseudo-Boolean algorithm performs backtracking search, making decisions, propagating the consequences of those decisions and inferring new constraints that prune the search space whenever the decisions performed lead to a conflict.

Given a PBO instance $(F, \mathcal{O})$, pure solution-improving search works by iteratively invoking a PB-oracle on the working formula $F \cup \{\mathcal{O} < \text{UB}\}$ consisting of the constraints F of the original PBO instance and (the normalized version) of the so-called *solution-improving constraint* $\mathcal{O} < \text{UB}$, which enforces a known upper bound on the minimum-cost of the PBO instance and thus restricts the search to solutions that have lower costs than UB. That is, the solution-improving constraint is satisfied by an assignment M if and only if $\mathcal{O}(M) < \text{UB}$; if the PB-oracle finds a solution M^w to the working formula, the algorithm improves on its currently best known solution. The solution-improving constraint is then iteratively strengthened to $\mathcal{O} < \mathcal{O}(M^w)$ and the PB-oracle invoked again. Assuming that an initial solution is found before a possible time limit, the algorithm can return the latest found solution at any time and can thus be considered an anytime approach. The algorithm terminates at the time limit or when the PB-oracle determines that there are no solutions to the current working instance. In the latter case, the last solution found is guaranteed to be of minimum cost for the input PBO instance. In other words, solution-improving search is a complete algorithm for PBO which can be implemented as a stand-alone solver as well as in combination with other solving approaches [32, 33, 12]. For the approach we develop in this work, the more important aspect of solution-improving search is its anytime feature; our approach integrates solution-improving search together with so-called oracle-based local search.

Example 2. Consider an invocation of pure solution-improving search on the PBO instance $(F, \mathcal{O})$ detailed in Example 1. The initial call to the PB-oracle `PB-Solve` is done on the working formula F (conceptually adding the solution-improving constraint $\mathcal{O} < \infty$). There are many possible solutions that the oracle can return; for the sake of this example, assume that the oracle returns the solution $M^* = \{b_1, b_2, b_3, \bar{b_4}, b_5\}$ which has cost $\mathcal{O}(M^*) = 17$. This solution is stored as the current best solution. In the next iteration, the PB-oracle is invoked on the formula $F \cup \{\mathcal{O} < 17\}$. There are again many solutions that could be returned; assume that the PB-oracle returns the solution $M^* = \{\bar{b_1}, \bar{b_2}, b_3, b_4, b_5\}$ with $\mathcal{O}(M^*) = 9$. In the next iteration, the oracle is invoked on $F \cup \{\mathcal{O} < 9\}$. Now the oracle determines that there are no solutions to the current working formula. Hence the algorithm terminates and returns the latest M^* as a minimum-cost solution.

3.2 Stochastic Local Search for PBO

The other approach to anytime pseudo-Boolean optimization which—as a general algorithmic paradigm—is relevant for our work, is that of stochastic local search (SLS) [19]. Starting from a heuristically chosen variable assignment M^w, an SLS algorithm for PBO

searches for a low-cost solution to a given PBO instance $(F, \mathcal{O})$ by heuristically flipping the values assigned by M^w until a solution for F is found. The algorithm then attempts to improve on the found initial solution by making local changes to the solution in a similar manner. This process is iterated until a termination criterion is reached, at which point the best solution found so-far is returned. The main challenge in developing SLS algorithms is designing practically effective heuristics for choosing which variables to flip, as well as for initializing assignments in order to find a solution with the lowest possible cost without getting stuck in a local minimum. Typical SLS approaches integrate both greedy moves to a best solution in the local neighborhood of the current solution, and random walk steps which—in order to escape local minima—allow for moving to neighboring solution of higher cost than the current solution. A recently developed pure SLS approach is detailed in [22]; we refer to the original work for the specifics of the approach. Note that a pure SLS approach does not typically guarantee finding an optimal solution regardless of how much resources are given, and such approaches do not integrate mechanisms which would allow for detecting whether a current solution is of minimum-cost. The oracle-based local search approach to PBO developed in this work, as detailed next, can be viewed either as a non-traditional form of local search in which a complete PB-oracle is used as the basis for moving from one solution to another and detecting when a minimum-cost solution is found, or as an extension of solution-improving search with local-search elements; the approach integrates aspects of both local search and solution-improving search.

4 ORACLE-BASED LOCAL SEARCH FOR PBO

In this section we detail `OraSLS`, the oracle-based local search algorithm for PBO developed in this paper. The algorithm draws ideas from both solution-improving search and stochastic local search. On a high level, `OraSLS` can be seen as a local search algorithm over complete assignments to the objective variables of a given PBO instance. However, in contrast to traditional local search, `OraSLS` works entirely with solutions to the input PBO instance. More specifically, whenever the value of an objective variable would be flipped, `OraSLS` relies on a PB-oracle to determine whether solutions to the input instance that extend the current assignment of objective variables actually exist. `OraSLS` goes beyond this type of non-traditional approach to stochastic local search by also integrating iterations of solution-improving search that are invoked whenever the oracle-based local search is unable to improve on the current solution; conceptually being stuck in a local optimum. This guarantees that, given enough resources, the search is bound to find a minimum-cost solution.

The PB Oracle. In the following detailed description of `OraSLS`, we abstract the use of the PB-oracle into the function `PB-Solve`. In contrast to solution-improving search, our algorithm often makes use of the oracle as an incomplete decision procedure by invoking it under a resource limit (in practice, limiting the number of conflicts seen before terminating the oracle call, as will be detailed later in Section 5). More specifically, given a formula F, a partial assignment $\mathcal{A}$ of its variables, and a resource limit α, the call `PB-Solve`$(F, \mathcal{A}, \alpha)$ runs a conflict-driven pseudo-Boolean decision solver until one of the following three scenarios realize:

(i) a solution $M \supset \mathcal{A}$ to F that extends $\mathcal{A}$ is found,
(ii) the decision solver determines that there is no such solution, or
(iii) the decision solver reaches the resource limit α at which point the decision solver is interrupted.

Note that setting $\alpha = \infty$ guarantees that scenario (iii) will never happen. For each of the scenarios, the decision solver returns a tuple $(M, \textit{sat?})$, where *sat?* is true if a solution $M \supset \mathcal{A}$ to F was found.

Algorithm 1: OraSLS: Oracle-based local search for PBO

Input: A PBO instance $\mathcal{I} = (F, \mathcal{O})$
Output: The best solution M^* to $\mathcal{I}$ found

```
1  init-polarities-and-activities()
2  (M, sat?) ← PB-Solve(F, ∅, ∞)
3  if not sat? then return "no feasible solutions"
4  F ← F ∪ {O < O(M*)};  M* ← M
5  while true do
6      A ← ∅
7      set-sticky-polarities(M*)
8      shuffled-obj-vars ← reorder(VAR(O))
9      for x ∈ shuffled-obj-vars do
10         if M(x) = 0 then A ← A ∪ {x̄}
11         else
12             (M, sat?) ← PB-Solve(F, A ∪ {x̄}, α)
13             if sat? then
14                 A ← A ∪ {x̄}
15                 if O(M) < O(M*) then M* ← M
16             else
17                 A ← A ∪ {x}
18                 if |A ∩ VAR(O)| > β then break
19     if stagnation then
20         F ← F ∪ {O < O(M*)}
21         (M, sat?) ← PB-Solve(F, ∅, ∞)
22         if not sat? then return M*
23         else M* ← M
24 return M*
```

Overview of Oracle-Based SLS for PBO

Algorithm 1 details `OraSLS` in pseudo-code. Invoked on an input PBO instance $(F, \mathcal{O})$, `OraSLS` first calls the PB-oracle without assumptions and without resource limits to check that the input instance has a solution (Line 2); otherwise the instance at hand would essentially not be a PBO instance but rather an unsatisfiable PB decision problem instance (in which case the algorithm will terminate early on Line 3). Invoked without assumptions (i.e., under an empty partial assignment) and without a resource limit, the PB-oracle is guaranteed to provide an initial solution if one exists. The initial solution is stored in M and in M^*. The latter M^* will for the duration of the search always be the lowest-cost solution found so-far. The algorithm then adds a solution-improving constraint to the PB formula F of the input instance, using the cost of the initial solution as the right-hand-side upper bound (Line 4).

After the just-described initialization phase, the main search loop (Lines 5-18) is entered. The loop is iterated until either (i) an optimal solution is found or (ii) the algorithm is stopped by the user (or using a pre-described limit) at which point it outputs the lowest-cost solution M^* found so far.

Each iteration of the main search loop begins by using the `reorder` function for determining an ordering of the objective vari-

ables, storing the ordering in *shuffled-obj-vars*, and initializing a partial assignment $\mathcal{A}$. The `reorder` function in a sense corresponds to an ordering in which a traditional SLS algorithm may flip the values assigned to individual variables. As we will see, the order of the variables in *shuffled-obj-vars* maps directly to the order in which the search will attempt to improve the current assignment M by modifying the current solution. Hence the implementation of the `reorder` function can be expected to play a key role in the performance of the approach. Our implementation of `reorder` is detailed in Section 5. At each point during an iteration, the partial assignment $\mathcal{A}$ consists of those objective variables that have already been processed during the iteration, fixed for the remainder of the iteration.

More specifically, after shuffling the objective variables via `reorder`, the inner loop (Lines 9 to 18) processes the objective variables in the obtained order. When processing a variable $x \in$ *shuffled-obj-vars*, the assignment of x under the current solution M is checked. If $M(x) = 0$, then the assignment of x does *not* currently incur cost on M. Informally speaking, in this case M cannot be locally improved by flipping the value of x. With this intuition, the value of x is fixed to 0 for the rest of the inner loop by adding $\bar{x}$ to $\mathcal{A}$ on Line 10. Otherwise (Line 11), the PB-oracle is invoked to check for the existence of a solution that agrees with the variables in $\mathcal{A}$ (the already-processed variables during this iteration) and the additional assumption that $x = 0$ (Line 12). If such a solution is obtained from the PB-oracle, the value of x is fixed to 0 for the rest of the inner loop on Line 14 and the solution M returned by the PB-oracle is compared to the currently best known solution M^* on Line 15. Otherwise, the value of x is fixed to 1 for the rest of the inner loop on Line 18.

Important for practical efficiency of the approach, the call to the PB-oracle on Line 12 is made under a resource limit α, in practice limiting how much time is spent in the PB-oracle. As such, fixing $x = 1$ within the inner loop can happen both if the PB-oracle determines that no solution extending $\mathcal{A} \cup \{\bar{x}\}$ exists, or when the PB-oracle is terminated due to having reached a resource limit. If the value of x is fixed to 1 during an iteration, the iteration is regarded as a *failed* attempt to extend $\mathcal{A}$ without incurring more cost. As detailed later on in Section 5, for practical performance it is also reasonable to limit the number of failed attempts by using a fail-limit β (Line 18). (Note that the number of elements in $\mathcal{A} \cap \text{VAR}(\mathcal{O})$ equals the number of objective variables fixed to 1 so far in this iteration.)

Example 3. Consider invoking `OraSLS` on the PBO instance $(F, \mathcal{O})$ from Example 1 and an iteration of the main loop in which the current solution M is $\{b_1, b_2, b_3, \bar{b_4}, b_5\}$. Assume that the `reorder` function returns *shuffled-obj-vars* $= \{b_1, b_4, b_2, b_3, b_5\}$. Then in the first step of the inner loop $\mathcal{A}$ is empty. Since $M(b_1) = 1$ the first query to the PB-oracle is `PB-Solve`$(F, \{\bar{b_1}\}, \alpha)$ under the assumptions $\{\bar{b_1}\}$. This constitutes asking the PB-oracle for any solution in which $b_1 = 0$. Assume that the solver returns $M = \{\bar{b_1}, b_2, b_3, b_4, \bar{b_5}\}$, which results in $\bar{b_1}$ being added to $\mathcal{A}$. In the next iteration, the variable to be processed is b_4. Since $M(b_4) = 1$—note that the solution changed in the first iteration—the PB-oracle is invoked on `PB-Solve`$(F, \{\bar{b_1}, \bar{b_4}\})$. Assume that this time the PB-oracle is unable to find a solution, and so b_4 is added to $\mathcal{A}$. The third variable to be processed is b_2. The PB-oracle is this time invoked on `PB-Solve`$(F, \{\bar{b_1} b_4, \bar{b_2}\})$. Now assume that the solver returns $M = \{\bar{b_1}, \bar{b_2}, b_3, b_4, b_5\}$, resulting in $\bar{b_2}$ being added to $\mathcal{A}$. In the last two iterations of the inner loop (corresponding to processing variables b_3 and b_4) no more solutions are found. Note that the instantiation of `reorder` can strongly influence the quality of solutions `OraSLS` finds. If the `reorder` function in Example 3 would have returned *shuffled-obj-vars* $= \{b_4, b_5, b_1, b_2, b_3\}$ instead, the inner loop could at best only have discovered the solution $\{b_1, b_2, b_3, \bar{b_4}, \bar{b_5}\}$.

5 SEARCH HEURISTICS

With the high-level description of OraSLS in place, we turn to discussing the various heuristic choices that the approach allows for. In particular, these deal with (i) the instantiation of `reorder` for ordering the objective variables at each iteration; (ii) how to limit resources on various level of the search; and (iii) how to guide the PB-oracle's internal search via altering the scores of it decisions heuristics in order to efficiently find initial solutions (before the main loop) and improving solutions (in the inner loop).

5.1 Ordering of Objective Variables

The instantiation of `reorder` is one of the central design choices in `OraSLS`, as it governs intensification and diversification in terms of how the solution space is traversed. We instantiate `OraSLS` as follows. For intensification, `OraSLS` employs what we call "shuffling". In shuffling, the objective variables are sorted by their coefficients in decreasing order. For integrating limited randomization into the intensification step, the sorted variables are then partitioned into n (a search parameter) equal-sized buckets (with a potentially smaller last bucket) that are then one-by-one randomly permuted. Shuffling is performed at every even iteration of the main loop. For diversification, `OraSLS` fully reverses at every odd iteration the current ordering of the objective variables.

5.2 Limiting Resources for Performance

We turn to the question of how to limit resources on various levels of the search in order to avoid the overall search getting stuck. For this, we employ resource limits on three levels: on the level of individual calls to the PB-oracle (to avoid the overall search getting stuck in potentially significantly hard individual PB-oracle calls—using a *conflict limit*), on the level of how much resources are used within individual iterations of the inner loop (using a *fail limit*), and on the level of search stagnation (not finding improving solutions) in the main search loop overall (using a *stagnation limit*);

Conflict Limit The goal of the PB-oracle call performed in the inner loop (on Line 12) is to quickly determine whether the current partial assignment $\mathcal{A}$ is extendable into a solution M to the PBO instance at hand. Some of these PB-oracle calls can take a significant amount of time, which is unacceptable for an anytime search procedure geared towards finding relatively good solutions fast. In particular, instead of insisting on using significant runtime to prove the non-existence (or to find) such a M, it is more sensible to terminate costly PB-oracle calls early and thereafter direct the search to another part of the search space. This is achieved by fixing the variable under consideration at the time to 1, i.e., *to incur cost*, similarly as what would be done if the PB-oracle would report that there are no solutions. In practice, as the resource bound for terminating PB-oracle early, we limit the number of allowed conflicts in each conflict-driven search PB-oracle call with the parameter α. The notable exceptions to this are the very first call in Line 2 for finding an initial solution, and the PB-oracle call after stagnation in Line 21, since early termination of the oracle at these points would not necessarily allow the algorithm to escape stagnation.

Fail Limit We found the ordering of the objective variables in the inner loop of `OraSLS` can have a very significant effect on whether new solutions improving on the best known solution M^* are found. If better solutions are time-consuming to find or do not exist, `OraSLS` can spend a significant amount of time in iteratively invoking the PB-oracle without making any progress (regardless of whether this is due to the oracle being interrupted due to the conflict limit or because of determining that no better solutions exist). In order to prevent this, we enforce a fail limit parameter β that limits the number of variables that can be processed without improving the currently best known solution within each iteration of the inner loop. Exceeding the limit results in terminating the current iteration of the inner loop early.

Stagnation Limit The search of `OraSLS` can stagnate at the level of the main search routine in case the inner loop is invoked consecutively many times without improving on the currently best known solution, i.e., without executing Line 15 at least once (regardless of why the inner loop is exited—due to exceeding the fail limit or otherwise). If stagnation occurs—indicated in the pseudo-code by the Boolean *stagnation* becoming true—the next time Lines 19-23 are reached, `OraSLS` enforces a solution-improving constraint to F and invokes `PB-Solve` on the formula without assumptions or conflict limit. As such the call will either find a solution that has cost lower than the current M^*, or determine that the current solution is in fact minimum-cost. In other words, stagnation prevention is what makes `OraSLS` complete. We use the δ parameter to represent the number of times the inner loop can be tried without improving on M^*, i.e., without executing Line 15 at least once before stagnation occurs. Note the difference between breaking the inner loop and stagnation prevention. The variable *stagnation* becomes true if δ different *orderings* of objective variables are processed without improving M^* while the inner loop is broken if β *variables* in the ordering have been fixed to 1, i.e., processed without improving the current solution.

5.3 Variable Activities and Polarities in the PB-Oracle

We also consider guiding the search within individual calls to the PB-Oracle specifically with the type of search `OraSLS` implements in mind, building on similar ideas employed in the context of the Mrs. Beaver approach to anytime MaxSAT [26]. The two specific search heuristics of a PB-oracle we consider are the activity heuristics of variables (which governs which variables are decided on next during search), and the default polarity selection for a variable (which governs to which value a variable is assigned to when making a decision on the variable). By default, variable activities in the PB-oracle are initialized uniformly, which means that the PB-oracle will in the beginning choose decision variables simply in the order in which the variables are in its internal data-structures. The activity is increased by the PB-oracle whenever a variable is involved in a conflict, alike in the central VSIDS decision heuristics in CDCL SAT solvers [25]. As for the variable polarities, they are initialized to 0 by default in the PB oracle, which means that the PB oracle decides to set the variables to 0 first. *Phase saving* [29], similarly as in CDCL SAT solvers, is by default employed: variable polarity selection heuristics for a variable x are updated whenever constraint propagation assigned a value v to x, so that next time x is decided on, it will be assigned to v.

As detailed next, we focus search in particular on objective variables, and employ a modified activity and polarity initialization that aims a finding better initial solutions faster, as well as replace phase saving with an alternative that takes into account the current best solution while the PB-oracle is trying to improve on it.

Oracle Heuristics for Initial Solutions For finding an initial solution, we force the PB-oracle to first make decisions on objective variables by increasing the activity scores of objective variables by one unit in the `init-polarities-and-activities` procedure on Line 1 of Algorithm 1. We also initialize variable polarities of objective variables *optimistically* so that deciding on an objective variables does not incur cost in terms of the objective function. Combining the two guides the search for an initial solutions of the PB-oracle toward finding a solution of relatively low cost, with ideally many of the objective variables assigned in a way that they do not incur cost.

Oracle Heuristics for Improving Solutions Once an initial solution is found, the inner loop (Lines 9-23 of Algorithm 1) aims to improve on it. For the PB-oracle in the inner loop, as an approach-specific form of phase saving, we set the polarities of the variables according to the polarities of variables in the current best solution. Furthermore, motivated by earlier work on oracle heuristics in anytime MaxSAT solving (going under the name solution-guided search in [10, 4] and "conservative, fix, original-variables" in [26]), we activate what we refer to as "sticky polarities" (Line 7). This works as follows. If the polarity of a variable x is set to v, then the PB-oracle will never assign $1 - v$ as a decision; with sticky polarities, the only way x can be assigned to $1 - v$ during the PB-oracle search is by constraint propagation. Intuitively, sticky polarities intensify search toward the currently best solution, this approach has been shown to provide performance improvements in the context of MaxSAT [26].

6 EMPIRICAL EVALUATION

We turn to an empirical evaluation of `OraSLS`.

6.1 Setup

The experiments we report on were run on computing nodes with 8-core Intel Xeon E5-2670 2.6-GHz CPUs and 64-GB RAM.

Implementation and Competing Solvers We implemented the approach in C++ building on the source code of RoundingSAT (commit: a7fe32d8) [15]. The open-source implementation and full empirical data are available via https://bitbucket.org/coreo-group/orasls/. RoundingSAT is a state-of-the-art pseudo-Boolean decision solver, and hence a natural choice for the PB-oracle. We evaluate the following variants of our approach.

OraSLS: Default version of our implementation of `OraSLS`. Default parameters based on preliminary evaluation are: conflict-limit 20, fail-limit 10, stagnation-limit of 1. The number of buckets (in the `reorder` function) is 8.

SIS: Baseline pure solution-improving search using the same source code as OraSLS, obtained by disabling line 1 and lines 7-19 in `OraSLS`.

SIS+: Another baseline in-between SIS and OraSLS, obtained by disabling lines 8-19 in `OraSLS` (and, in contrast to SIS, lines 1 and 7 enabled). That is, SIS+ is a refinement of SIS which employs the same activity and polarity initialization and sticky phases heuristics as OraSLS.

The values for the parameters used by **OraSLS** in these experiments are based on small-scale "by hand" experimentation performed during development. The values were not exhaustively optimized by employing, e.g., an automatic configurator.

Furthermore, we compare the performance of OraSLS to the following recent anytime and complete PBO solvers.

LS-PBO [22], using its default parameters, is a recently-proposed pure stochastic local search solver for PBO, as the main non-complete approach to compare to.

RoundingSAT includes an implementation of a solution-improving search algorithm (**LSU**) [14] as well as a recent exact hybrid approach that combines core-guided optimization (**OLL**, sometimes referred to as MSU4) with LSU [12]. In order to obtain anytime solutions from RoundingSAT, we modified it to output an objective value upper bound whenever it finds a better solution during search. We include the LSU as well as OLL implementations of this PBO solver in our experiments as a natural choice, since OraSLS employs RoundingSAT as its decision PB-oracle.

Performance Metric In contrast to complete (exact) solvers, which are typically compared against one another in terms of how long each solver takes to find a provably optimal solution, comparing anytime solvers requires different types of metrics. Here we use the established performance metric for ranking anytime solvers used in MaxSAT Evaluation [2] (MSE), the main competitive event for MaxSAT solvers. For our empirical evaluation, the set of solvers is $\mathcal{S} = \{$OraSLS, SIS, SIS+, LS-PBO, RoundingSAT LSU, RoundingSAT OLL$\}$. For a solver $s \in \mathcal{S}$ and instance in $\mathcal{I}$ let $\text{BEST-COST}(\mathcal{I}, s)$ be the minimum cost of the solutions found by s when invoked on $\mathcal{I}$ in our experiments and $\text{BEST-COST}(\mathcal{I}) = \min_{s \in \mathcal{S}}\{\text{BEST-COST}(\mathcal{I}, s)\}$ the minimum cost of all solutions found by any solver. The MSE score $\text{SCORE}(\mathcal{I}, s)$ of s on $\mathcal{I}$ is then 0 if s was unable to find any solution of s within the time limit and:

$$\text{SCORE}(\mathcal{I}, s) = \frac{\text{BEST-COST}(\mathcal{I}) + 1}{\text{BEST-COST}(\mathcal{I}, s) + 1}$$

otherwise. The MSE score takes values between 0 and 1 and—intuitively speaking—measures how close to the best-known solution the solution computed by a solver is. A score value of 1 means that the solver was able to compute a solution whose cost matches the best known solution and a value of 0.5 means that the found solution has twice the cost of the best known solution. We enforced a per-instance time limit of 5 minutes and a memory limit of 16 MB on each solver, following the incomplete track of MaxSAT Evaluations.

Benchmarks We consider the same heterogeneous set of benchmarks, with 50 different benchmark domains, as used recently in benchmarking complete PBO solvers [33]. The full set of benchmarks contains optimization benchmarks used in Pseudo-Boolean Competition 2016 [30] (as the most recent instantiation of the competition) and 0-1 integer programs from the MIPLIB library [16]. Following [33], filtering was applied to remove all unsatisfiable benchmarks, benchmarks without objective functions and benchmarks with very large ($\geq 2^{64}$) coefficients. To obtain a balanced set of benchmark in terms of the various different benchmark domains, following [32, 33] we randomly sampled 20 instances from each benchmark domain, including all instances if the family had 20 or less instances; as shown in [32], such a random sampling of benchmarks allows for obtaining robust sample results when comparing different solvers. After the sampling, we ended up with a benchmark set consisting of a total of 865 instances.

6.2 Results

We first compare the MSE scores of OraSLS against those of LS-PBO, RoundingSAT LSU and RoundingSAT OLL. Starting with LS-PBO as state-of-the-art pure stochastic local search style competitor of OraSLS, Table 1 shows the average MSE score of OraSLS and LS-PBO for the 24 benchmark domains on which the domain-specific average score of the two solvers differs by at least 0.1. OraSLS wins LS-PBO on 18 of the 24 domains. OraSLS is significantly better in providing solutions, with timeouts (TO, i.e., no solutions found) on a total on 89 instances against 194 for LS-PBO, which underlying the benefit of using a PB-oracle. Complementing the tabulated results, Figure 1 (left plot) provides a visualization of the per-instance MSE scores of these two solvers.

Table 1. OraSLS vs LS-PBO: Average MSE scores for domains on which the score differ by ≥ 0.1

		LS-PBO		OraSLS	
Benchmark domain	**#**	**Score**	**#TO**	**Score**	**#TO**
MANETS	20	0.19	16	**0.87**	2
airplane-cost-quality	20	0.80	4	**1.00**	0
decomp	10	0.88	0	**0.99**	0
domset	15	**1.00**	0	0.85	0
golomb-rulers	20	0.52	8	**0.84**	3
haplotype-inf	20	**0.99**	0	0.77	0
lecture-timetabling	20	0.31	0	**0.93**	0
logic-synthesis	20	**1.00**	0	0.89	0
market-split	20	0.19	11	**0.35**	11
miplib-neos	20	0.67	5	**0.81**	3
miplib-various	20	0.48	10	**0.84**	2
netlib-various	20	0.43	11	**0.58**	7
number-factorization	20	0.20	16	**1.00**	0
plan-museum-visits	20	0.40	12	**0.89**	2
prime-implicants	20	0.55	9	**0.75**	5
radar-station-alloc	12	**1.00**	0	0.16	10
repair-bionet	20	0.82	0	**1.00**	0
transport-systems	20	0.89	0	**1.00**	0
transportation	20	0.20	16	**0.71**	4
unibo-various	20	0.18	15	**0.77**	4
upgradability	20	0.76	1	**1.00**	0
vm-workload	20	**0.60**	8	0.44	11
wnq	16	**1.00**	0	0.42	0
workshift-design	20	0.12	17	**0.98**	0
All domains	865	0.73	194	**0.84**	89
Wins (#domains)			6		**18**

Table 2 and Figure 1 (right) provide a comparison of OraSLS against the anytime versions of RoundingSAT's LSU and OLL implementations. In terms of average MSE scores, both RoundingSAT variants come closer to OraSLS than LS-PBO, with OLL being the closest; this is also reflected in the average score of the solver over all benchmarks (see Table 4). However, for domains where the difference in scores between any two of the three solvers is ≥ 0.1, which is the case for 8 benchmark domains, OraSLS outperforms both RoundingSAT variants on 7 of these domains. Furthermore, the differences appear quite significant, for example, with differences ≥ 0.37 for the course-alloc domain in the benefit of OraSLS.

Finally, we consider in more detail the performance of the fully-fledged OraSLS against its pure SIS variants and the in-between variant SIS+. Table 3 provides the comparison for the 14 benchmark domains on which the average MSE scores of the best and worst performance variants on the domain differ by at least 0.1. These results more importantly show that the combination of techniques in the fully-fledged OraSLS pay off in terms of performance, with OraSLS winning on 9 domains against 3 and 1 domains, respectively,

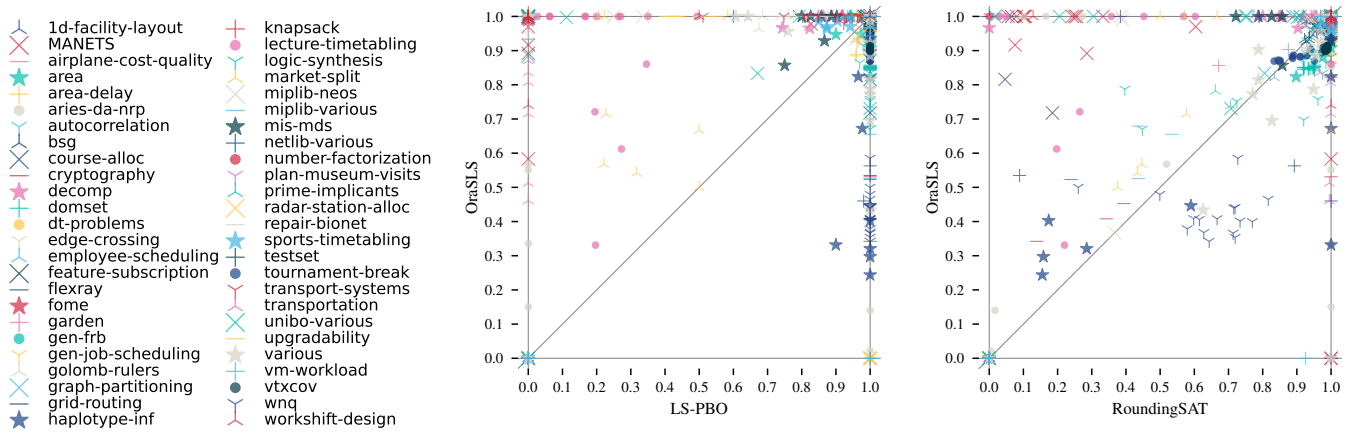

Figure 1. Per-instance performance comparison. Left: OraSLS vs LS-PBO. Right: OraSLS vs RoundingSAT OLL.

Table 2. OraSLS vs RoundingSAT LSU and RoundingSAT OLL: Average MSE scores for domains on which the largest difference in scores ≥ 0.1

		RS LSU		RS OLL		OraSLS	
Benchmark domain	#	**Score**	**#TO**	**Score**	**#TO**	**Score**	**#TO**
MANETS	20	0.45	0	0.52	0	**0.87**	2
aries-da-nrp	20	0.44	1	0.61	1	**0.68**	2
course-alloc	6	0.54	0	0.55	0	**0.92**	0
decomp	10	0.33	6	0.82	1	**0.99**	0
lecture-timetabling	20	0.85	0	0.69	0	**0.93**	0
unibo-various	20	0.59	4	0.61	4	**0.77**	4
wnq	16	0.53	0	**0.65**	0	0.42	0
workshift-design	20	0.53	0	0.96	0	**0.98**	0
All domains	865	0.80	90	0.83	83	0.84	89
Wins (#domains)			0		1		**7**

for SIS+ and SIS; this includes a tie on 2 domains for OraSLS and SIS+. Furthermore, on the only domain on which SIS+ wins OraSLS, the scores of these two solvers differ only by 0.01.

Table 3. OraSLS, SIS and SIS+: Average MSE scores for domains on which the largest difference in scores ≥ 0.1

		SIS		SIS+		OraSLS	
Benchmark domain	#	**Score**	**#TO**	**Score**	**#TO**	**Score**	**#TO**
MANETS	20	0.45	0	0.74	2	**0.87**	2
aries-da-nrp	20	0.44	1	0.63	2	**0.68**	2
course-alloc	6	0.54	0	**0.93**	0	0.92	0
cryptography	11	0.59	0	0.66	0	**0.73**	0
decomp	10	0.33	6	0.98	0	**0.99**	0
haplotype-inf	20	0.61	0	0.74	0	**0.77**	0
repair-bionet	20	0.00	0	**1.00**	0	**1.00**	0
unibo-various	20	0.59	4	0.73	4	**0.77**	4
upgradability	20	0.31	0	**1.00**	0	**1.00**	0
wnq	16	**0.53**	0	0.41	0	0.42	0
workshift-design	20	0.53	0	0.94	0	**0.98**	0
All domains	865	0.76	90	0.83	89	0.84	89
Wins (#domains)			1		3		**9**

As a high-level overview, the average MSE scores of OraSLS, its baseline versions SIS and SIS+, and the two competing approaches over *all* benchmark instances are shown in Table 4. Again, we observe that OraSLS outperforms all of the competing approaches also when considering all benchmarks as a single set. Interestingly, the performance of the recent LS-PBO approach implementing pure stochastic local search exhibits the weakest performance and is also outperformed by RoundingSAT when run as an anytime solver.

Table 4. Average MSE scores over *all* benchmark domains

Solver	**Score**	**#TO**
OraSLS	0.8422	89
SIS+	0.8316	89
RS OLL	0.8299	83
RS LSU	0.8049	90
SIS	0.7597	90
LS-PBO	0.7280	194

7 CONCLUSIONS

Motivated by the recent success of new types of anytime approaches to MaxSAT solving, we studied the applicability of the oracle-based local search approach in the more general context of pseudo-Boolean optimization. Overall, in the context of PBO, the OraSLS approach naturally allows for integrating both pure oracle-based transitions from one solution to another and solution-improving search, with various heuristic options to vary search behavior. Empirically, the approach turned out to be promising, outperforming on various benchmark domains clearly the recent pure stochastic local search approach, the solution-improving search implementation of RoundingSAT, as well as often also the core-guided RoundingSAT approach as an anytime solver. Since the best-performing approaches in the evaluation turned out to be OraSLS and anytime core-guided RoundingSAT, a promising direction for future work is to study various of combining aspects of both these approaches towards further performance improvements.

ACKNOWLEDGMENTS

Work financially supported by Academy of Finland grants 322869, 342145, and 356046. The authors thank the Finnish Computing Competence Infrastructure (FCCI) for computational and data storage resources.

References

[1] Carlos Ansótegui and Joel Gabàs, 'WPM3: An (in)complete algorithm for weighted partial MaxSAT', *Artificial Intelligence*, **250**, 37–57, (2017).

[2] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins, 'MaxSAT Evaluation 2018: New developments and detailed results', *J. Satisf. Boolean Model. Comput.*, **11**(1), 99–131, (2019).

[3] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins, 'Maximum satisfiability', in *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 929–991, IOS Press, (2021).

[4] Jeremias Berg, Emir Demirovic, and Peter J. Stuckey, 'Core-boosted linear search for incomplete maxsat', in *CPAIOR*, volume 11494 of *Lecture Notes in Computer Science*, pp. 39–56. Springer, (2019).

[5] Daniel Le Berre and Anne Parrain, 'The Sat4j library, release 2.2', *J. Satisf. Boolean Model. Comput.*, **7**(2-3), 59–6, (2010).

[6] *Handbook of Satisfiability - Second Edition*, eds., Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021.

[7] Shaowei Cai and Zhendong Lei, 'Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability', *Artif. Intell.*, **287**, 103354, (2020).

[8] Shaowei Cai, Xindi Zhang, Mathias Fleury, and Armin Biere, 'Better decision heuristics in CDCL through local search and target phases', *J. Artif. Intell. Res.*, **74**, 1515–1563, (2022).

[9] Francisco Chicano, L. Darrell Whitley, and Renato Tinós, 'Efficient hill climber for constrained pseudo-boolean optimization problems', in *GECCO*, pp. 309–316. ACM, (2016).

[10] Emir Demirovic and Peter J. Stuckey, 'Techniques inspired by local search for incomplete maxsat and the linear algorithm: Varying resolution and solution-guided search', in *CP*, volume 11802 of *Lecture Notes in Computer Science*, pp. 177–194. Springer, (2019).

[11] Jo Devriendt, Ambros Gleixner, and Jakob Nordström, 'Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search', *Constraints*, (2021).

[12] 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 *AAAI*, pp. 3750–3758. AAAI Press, (2021).

[13] 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).

[14] Jan Elffers and Jakob Nordström, 'Divide and conquer: Towards faster pseudo-Boolean solving', in *IJCAI*, pp. 1291–1299. ijcai.org, (2018).

[15] Jan Elffers and Jakob Nordström, 'A cardinal improvement to pseudo-boolean solving', in *AAAI*, pp. 1495–1503. AAAI Press, (2020).

[16] 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).

[17] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure, 'A CSP solver focusing on fac variables', in *CP*, volume 6876 of *Lecture Notes in Computer Science*, pp. 493–507. Springer, (2011).

[18] John N. Hooker, 'Generalized resolution for 0-1 linear inequalities', *Ann. Math. Artif. Intell.*, **6**(1-3), 271–286, (1992).

[19] Holger H. Hoos and Thomas Stützle, 'Stochastic local search', in *Handbook of Approximation Algorithms and Metaheuristics (1)*, 297–307, Chapman and Hall/CRC, (2018).

[20] Holger H. Hoos and Edward P. K. Tsang, 'Local search methods', in *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, 135–167, Elsevier, (2006).

[21] Zhendong Lei and Shaowei Cai, 'NuDist: An efficient local search algorithm for (weighted) partial MaxSAT', *Comput. J.*, **63**(9), 1321–1337, (2020).

[22] Zhendong Lei, Shaowei Cai, Chuan Luo, and Holger H. Hoos, 'Efficient local search for pseudo boolean optimization', in *SAT*, volume 12831 of *Lecture Notes in Computer Science*, pp. 332–348. Springer, (2021).

[23] Chu Min Li and Felip Manyà, 'MaxSAT, hard and soft constraints', in *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 903–927, IOS Press, (2021).

[24] João Marques-Silva, Inês Lynce, and Sharad Malik, 'Conflict-driven clause learning SAT solvers', in *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 133–182, IOS Press, (2021).

[25] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik, 'Chaff: Engineering an efficient SAT solver', in *DAC*, pp. 530–535. ACM, (2001).

[26] Alexander Nadel, 'Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization', in *FMCAD*, pp. 193–202. IEEE, (2019).

[27] Alexander Nadel, 'Anytime algorithms for MaxSAT and beyond', in *FMCAD*, p. 1. IEEE, (2020).

[28] Alexander Nadel, 'Polarity and variable selection heuristics for SAT-based anytime MaxSAT', *J. Satisf. Boolean Model. Comput.*, **12**(1), 17–22, (2020).

[29] Knot Pipatsrisawat and Adnan Darwiche, 'A lightweight component caching scheme for satisfiability solvers', in *SAT*, volume 4501 of *Lecture Notes in Computer Science*, pp. 294–299. Springer, (2007).

[30] Olivier Roussel. Pseudo-Boolean Competition 2016. http://www.cril.univ-artois.fr/PB16/. Accessed: 25 April 2023.

[31] João P. Marques Silva and Karem A. Sakallah, 'GRASP - a new search algorithm for satisfiability', in *ICCAD*, pp. 220–227. IEEE Computer Society / ACM, (1996).

[32] Pavel Smirnov, Jeremias Berg, and Matti Järvisalo, 'Pseudo-boolean optimization by implicit hitting sets', in *CP*, volume 210 of *LIPIcs*, pp. 51:1–51:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2021).

[33] Pavel Smirnov, Jeremias Berg, and Matti Järvisalo, 'Improvements to the implicit hitting set approach to pseudo-boolean optimization', in *SAT*, volume 236 of *LIPIcs*, pp. 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2022).

[34] Renato Tinós, L. Darrell Whitley, and Francisco Chicano, 'Partition crossover for pseudo-boolean optimization', in *FOGA*, pp. 137–149. ACM, (2015).

[35] Joachim P. Walser, 'Solving linear pseudo-boolean constraint problems with local search', in *AAAI*, pp. 269–274. AAAI Press, (1997).