Improving Exact Algorithm for Pseudo Boolean Optimization with Two New Phase Selection Heuristics

Yujiao Zhao[1], Yizhan Xiang[1], Jiangnan Li[1], Yiyuan Wang[1,2,*], Minghao Yin[1,2]

[1] School of Information Science and Technology, Northeast Normal University, China
[2] Key Laboratory for Applied Statistics of Ministry of Education, Northeast Normal University, China
{zhaoyj, xiangyizhan, lijn101, wangyy912, ymh}@nenu.edu.cn

Abstract

Pseudo-Boolean optimization (PBO) problem involves optimizing a linear objective function under linear inequality constraints defined over Boolean variables. PBO is widely used for modeling many combinational optimization problems, particularly in some real-world scenarios. In core-guided CDCL-based exact solvers, the way branching variables are assigned, known as phase selection, significantly affects the solving efficiency. This paper introduces two strategies to enhance solver performance by improving phase selection. Firstly, we design a new phase selection strategy that actively guides variables in the objective function toward assignments closer to the optimal solution. Secondly, to prevent the solver from becoming trapped in local solutions, we propose a reinforcement learning-based rephase mechanism that dynamically updates and resets variable phases. We integrate two phase selection strategies into two state-of-the-art PBO solvers and compare them against top-performing solvers from the PB competitions, using benchmarks from these competitions for assessment. The experimental results show that our solvers outperform the winning solver from the competitions.

Code —
https://github.com/yiyuanwang1988/AAAI26-PBOcodes

Introduction

The Boolean satisfiability problem (SAT) is the first proven NP-complete problem (Cook 1971). With significant advances in SAT and its optimization problem maximum satisfiability (MaxSAT), their high-performance solvers are now widely applied to solving lots of real-world problems (Biere, Heule, and van Maaren 2021). However, in many cases, these encodings lead to very large problem instances, primarily due to the limited expressive power of conjunctive normal form (CNF) (Buss and Nordström 2021). Pseudo-Boolean (PB) constraints, which are linear inequalities over binary variables with integer coefficients, offer a natural and efficient way to model a wide range of NP-hard real-world problems (Biere, Heule, and van Maaren 2021).

This paper focuses on solving pseudo-Boolean optimization (PBO) problem, which involves optimizing a linear objective function subject to linear inequality constraints defined over Boolean variables. PBO is widely used for modeling many combinatorial optimization problems, especially in practical scenarios such as various scheduling problems (Lester 2021, 2022).

Existing algorithms for solving PBO can be broadly classified into local search (LS) algorithms and exact algorithms. LS algorithms are heuristic methods that start from an initial solution and iteratively explore its neighborhood to find improved solutions. Recent years have witnessed significant progress through advancements in heuristic design (Lei et al. 2021; Chu et al. 2023; Zhou et al. 2023; Jiang et al. 2024; Chen, Lei, and Lu 2024; Chen et al. 2024; Zhao et al. 2025). Exact algorithms for solving PBO mainly fall into two categories: core-guided algorithms and implicit hitting set (IHS) algorithms. Core-guided algorithms are built upon conflict-driven clause learning (CDCL)-based PB solver, with RoundingSat (Elffers and Nordström 2018; Devriendt et al. 2021) as a representative. Subsequent work has primarily focused on improving its key components (Devriendt 2020; Berre et al. 2020; Devriendt, Gleixner, and Nordström 2021; Le Berre and Wallon 2021; Devriendt 2024; Nieuwenhuis et al. 2024; Lomis et al. 2025; Nieuwenhuis et al. 2025). IHS-based approaches, such as PBO-IHS (Smirnov, Berg, and Järvisalo 2021, 2022; Ihalainen et al. 2025), combine a PB solver for core extraction with an integer programming solver for hitting set computation.

This work focuses on core-guided solvers based on the CDCL framework. In CDCL, the synergy among various components contributes to the overall efficiency of the solver. Among them, the phase selection heuristic plays a particularly important role, as the assignment of branching variables directly determines the subsequent propagation path. A commonly adopted strategy in PBO solvers is solution-based phase saving (Demirovic and Stuckey 2019), which assigns branching variables based on the best-known solution, thereby guiding the search toward its surrounding region. However, this method tends to overly focus the search around the neighborhood of the current best solution, potentially overlooking more promising regions, especially when the current solution is far from the global optimum.

To address the above issues, we propose two new

*Corresponding author.

phase selection strategies. Firstly, we design a phase selection method called relaxed target-optimum-rest-conservative (RTORC). RTORC combines an optimistic assignment strategy with a conflict-frequency scoring mechanism to improve the objective function value while avoiding excessive conflicts. Secondly, we propose a reinforcement learning-based rephase (RLRP) strategy, which updates variable assignment preferences by learning from historical solutions and resets phases upon restart, increasing search diversity and encouraging exploration of high-quality solution spaces.

We implement two phase selection strategies in two state-of-the-art exact PBO solvers, RoundingSat (Devriendt et al. 2021) and Exact (Devriendt 2024). In addition, LS is known for its ability to quickly converge to high-quality solutions. Based on this property, we integrate LS into the initial stage of the two exact solvers to quickly obtain a high-quality upper bound. We evaluate two exact solvers against the top-performing solvers from the PB competitions, using benchmarks from these competitions for assessment. Experimental results show that our two solvers outperform three dedicated PBO solvers in both solution count and solution quality, and are competitive with the mixed integer programming (MIP) solver.

Preliminaries

A PB constraint is a linear inequality constraint defined over Boolean variables, where the variables x_i take values of 0 (false) or 1 (true). A literal ℓ_i over a Boolean variable x_i is either x_i or $\overline{x}_i = 1 - x_i$. Since all linear inequality constraints can be converted into standard form, only the standard form is presented here.

$$\sum_i a_i \ell_i \geq b \tag{1}$$

where all the literals ℓ_i are distinct, and all the coefficients a_i and the *degree* b are non-negative integers. A cardinality constraint is a special type of PB constraint, where the coefficients a_i are all 1, i.e., $\sum_i \ell_i \geq b$.

A PB formula is a conjunction of PB constraints, denoted as $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where C_p $(p \in \mathbb{Z}, 1 \leq p \leq m)$ is a PB constraint. The PBO problem consists of a PB formula F and an objective function $O : \sum_i c_i \ell_i$ where c_i is an integer coefficient.

For a formula F, let $|F|$ denote the number of constraints in F. The sets of variables and literals appearing in F are written as $\text{Var}(F)$ and $\text{Lit}(F)$, and the same notation applies to O. $|\text{Var}(F)|$ and $|\text{Lit}(F)|$ represents the number of variables and literals, respectively. In addition, let $|C|$ denote the number of variables in constraint C. Variables and literals that appear in the objective function are referred to as objective variables and objective literals, respectively. Otherwise, they are referred to as non-objective variables and non-objective literals.

An assignment α is a function mapping literals to false or true, denoted $\alpha(x_i) = \ell_i$ or $\bar{\ell}_i$. We denote $|\alpha|$ as the length of α. If $|\alpha| = |\text{Var}(F)|$, it indicates a complete assignment. The goal of the PBO problem is to find a complete assignment α that satisfies all PB constraints and minimizes the objective function value, denoted as $obj(\alpha)$.

Overview of the General Framework

RoundingSat and Exact are two advanced exact PBO solvers that alternate between core-guided search and linear search. We incorporate LS module and two phase selection strategies into both solvers, resulting in the enhanced versions RoundingSat-PS and Exact-PS. The following describes the general framework shared by both solvers.

Algorithm 1 presents the pseudocode of the framework. We begin by introducing key data structures and associated flags. Let α denote the current assignment, α' and α^* represent the last and current best solutions. α_{ls} denotes the assignment obtained by LS. All of them are initially empty. The flag cg is set to false for linear search and true for core-guided search, as these two search methods employ different phase selection strategies. The flag $newSol$, initially set to false, indicates whether a new best solution has been found.

After initialization (Lines 1–2), the LS algorithm is first invoked (Lines 3–5), followed by CDCL-based search (Lines 6–34). If the upper bound UB equals the lower bound LB, the optimal solution has been found. Otherwise, the solver enters the search loop. In each iteration, a branching variable is selected and assigned (Lines 8–13), followed by unit propagation (Line 14). Two cases are then considered: If a conflict C_{confl} occurs (Lines 15–23), the handling depends on whether it arises under the assumption set. If not (i.e., linear search), standard CDCL conflict analysis and backtracking are applied (Lines 16–17 and 23). If the conflict is under assumptions, an unsatisfiable core is extracted to improve LB and reformulate the objective function (Lines 18–23). This is the key idea of core-guided search. If no conflict occurs (Lines 24–33), the solver checks whether a feasible solution has been found and proceeds to the next iteration (Lines 25–29). Additionally, if the restart condition is met, the solver backtracks to level 0 (Lines 30–33).

We next focus on the LS module and two phase selection strategies.

• LS[1] is invoked for two main purposes: (1) to quickly find a feasible solution and establish an early UB, and (2) to provide initial variable assignments for phase selection. Therefore, if a feasible solution is found, UB is updated, α_{ls} is recorded as the current best assignment α^* and $newSol$ is set to true (Lines 3–5).

• In RoundingSat and Exact, core-guided and linear search adopt different phase selection strategies. The main difference between the two lies in whether reasoning is performed under assumptions: an empty assumption set triggers linear search, while a non-empty set triggers core-guided search, i.e., $cg = true$ (Lines 8–9). Core-guided search uses the phase saving strategy (denoted ps) (Pipatsrisawat and Darwiche 2007; Sellmann and Ansótegui 2006), which caches the current assignment of variables before they are unassigned during backtracking (Lines 23 and 31). In contrast, linear search adopts solution-based phase saving (Demirovic and Stuckey 2019). In this work, we replace the solution-based phase saving strategy with the proposed RTORC strat-

[1]We adopt *NuPBO-DeepOpt+* (Zhao et al. 2025), which is specifically designed for PBO solving.

Algorithm 1: The general framework

Input : PB formula F and objective function O
Output: UB

1 $\alpha := \alpha' := \alpha^* := \alpha_{ls} := \emptyset$, $LB := 0$, $UB := \infty$;
2 $cg := newSol := false$, $O' := O$;
3 $\alpha_{ls} := LS(F, O)$;
4 **if** α_{ls} *is a feasible solution* **then**
5 $\quad UB := obj(\alpha_{ls})$, $\alpha^* := \alpha_{ls}$, $newSol := true$;
6 construct the assumption set $A \subseteq Lit(O')$;
7 **while** $UB - LB > 0$ **do**
8 $\quad$ **if** $A \neq \emptyset$ **then**
9 $\quad\quad cg := true$;
10 $\quad\quad$ choose $\ell_i \in A$, $\alpha(x_i) := \bar{\ell}_i$, $A := A \setminus \{\ell_i\}$;
11 $\quad$ **else**
12 $\quad\quad x_b := PickBranchVar()$;
13 $\quad\quad \alpha(x_b) := cg \,?\, ps(x_b) : RTORC(x_b, \alpha^*)$;
$\quad\quad$ `// RTORC is introduced in Sec.4.`
14 $\quad < \alpha, C_{confl} >:= UnitPropagation(F, \alpha)$;
15 $\quad$ **if** $C_{confl} \neq \emptyset$ **then**
16 $\quad\quad < bl, C_{learn} >:= ConflAnalysis(C_{confl}, \alpha)$;
17 $\quad\quad F.add(C_{learn})$;
18 $\quad\quad$ **if** *unsatisfiable under assumptions* **then**
19 $\quad\quad\quad LB := ImproveBound(O', C_{confl})$;
20 $\quad\quad\quad O' := Reformulate(O')$;
21 $\quad\quad\quad bl := 0, cg := false$;
22 $\quad\quad\quad$ construct the assumption set $A \subseteq Lit(O')$;
23 $\quad\quad BackTrack(F, \alpha, bl, ps)$;
24 $\quad$ **else**
25 $\quad\quad$ **if** $|\alpha| == |Var(F)|$ **then**
26 $\quad\quad\quad UB := obj(\alpha)$;
27 $\quad\quad\quad F.add(O < UB)$;
28 $\quad\quad\quad \alpha^* := \alpha, newSol := true, cg := false$;
29 $\quad\quad\quad$ construct the assumption set $A \subseteq Lit(O')$;
30 $\quad\quad$ **if** *time to restart* **then**
31 $\quad\quad\quad BackTrack(F, \alpha, 0, ps)$;
32 $\quad\quad\quad ps := RLRP(newSol, \alpha', \alpha^*)$; `// ps is disturbed by RLRP in Sec.5.`
33 $\quad\quad\quad \alpha' := \alpha^*, newSol := false$;
34 **return** UB;

egy (Line 13). Upon each restart, the proposed RLRP strategy is used to update ps (Line 32).

A New Phase Selection Strategy

This section first reviews the target-optimum-rest-conservative (TORC) phase selection strategy (Nadel 2019) used in MaxSAT. We then analyze its limitations in PBO and propose a relaxed version, called RTORC.

Review of the TORC Strategy

The TORC strategy is primarily applied in the anytime weighted MaxSAT algorithm and has demonstrated effective solving performance. The rules of TORC are described as follows (expressed in PB notation).

- **Optimistic strategy.** If the selected variable is an objective variable, an optimistic strategy is applied by setting its phase to the negation of the literal, encouraging all objective literals to be false.
- **Conservative strategy.** If the selected variable is a non-objective variable, a conservative strategy is applied by setting its phase to the value in the current best solution.

TORC is designed to combine the strengths of both conservative and optimistic strategies. On one hand, conservative heuristic has been shown to efficiently identify improved solutions by exploring the neighborhood of previous solutions (Demirovic and Stuckey 2019). On the other hand, by guiding the assignments of objective variables towards their ideal values, TORC aims to accelerate the convergence towards the optimal solution.

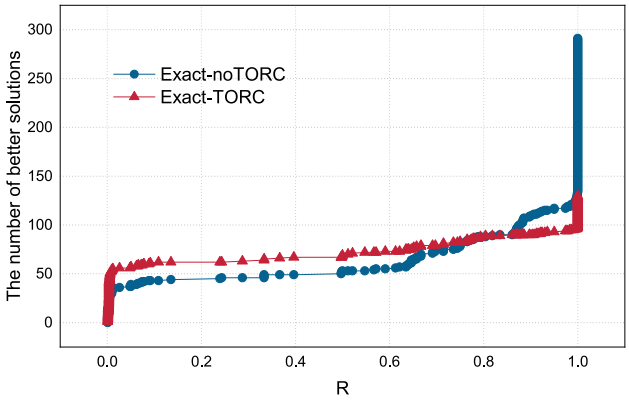

Figure 1: Solver performance comparison across R.

Preliminary Experimental Analysis

To evaluate the effectiveness of the TORC in PBO, we integrate it into Exact and conduct preliminary experiments.

Experimental Setup. The experiments are conducted on 1600 optimization instances from the PB competition 2016, with a time limit of 3600 seconds per instance. The modified solver is referred to as Exact-TORC, while the original baseline solver is denoted as Exact-nonTORC.

Experimental Analysis. We use the value of objective function to evaluate solver performance. Experimental results demonstrate that Exact-TORC performs significantly worse, with degradation on 291 instances and improvement on only 130 instances. We observe that performance degradation mainly occurs in instances where most or all variables in the PB constraints are objective variables, and such constraints are numerous. Satisfying such constraints often requires assigning some objective literals to true. If an overly optimistic strategy is used (i.e., assigning all objective literals to false), many constraints may be unsatisfied, causing frequent conflicts, excessive constraint learning, and frequent backtracking, which degrades solving efficiency.

In contrast, for instances with few objective variables that are sparsely distributed across the constraints, the solver can correct incorrect assignments with only a small number of conflicts, even if some objective literals need to be set to true. In such cases, the optimistic strategy does not cause

significant negative effects. Instead, it helps guide the search toward more promising regions in the early stages.

Figure 1 further supports our analysis. We introduce a simple ratio metric R, defined as the number of objective variables divided by the total number of variables. In Figure 1, the x-axis shows R, and the y-axis shows the number of instances where each solver performs better within each R interval. When R is small, indicating sparse distribution of objective variables, Exact-TORC outperforms Exact-nonTORC. As R increases, its advantage diminishes. At $R = 0.5$, the performance gap narrows, and at $R = 1$, Exact-nonTORC significantly outperforms Exact-TORC.

Potential Conflict Intensity

The metric R that only considers the total number of objective variables is too coarse. For example, suppose x_1 to x_5 are all objective variables. Consider two constraints: $C_1 : x_1+x_2+x_3+x_4+x_5 \geq 3$, and $C_2 : x_1+x_2+x_3+x_4+x_5 \geq 1$. If the branching variable x_1 is assigned false under the optimistic strategy, and x_2, x_3 are propagated to false by other constraints, then C_1 triggers a conflict, while C_2 may still be satisfied. This shows that even with the same number of objective variables, different constraint structures can vary in how favorable they are to the optimistic strategy. Therefore, we introduce a fine-grained metric called potential conflict intensity (PCI) to capture how likely a constraint is to trigger conflicts when assigning values to objective variables.

Algorithm 2 outlines the procedure for computing the PCI of a PBO instance. We first introduce two key operations for modifying PB constraints, including the $weakening$ rule and the $round2card$ rule. Consider a PB formula F, where each PB constraint C_p typically involves both objective variables and non-objective variables.

- **weakening rule:** By weaken$(C_p, \ell_j) \equiv \sum_{i \neq j} a_i \ell_i \geq b - a_j$, the rule removes a literal ℓ_j from a constraint by subtracting its corresponding coefficient a_j from degree b, resulting in a weaker but semantically valid constraint.
- **round2card rule:** This rule transforms a PB constraint C_p into a cardinality constraint C_p^{card}, where the degree b is set to the minimum number of literals that must be assigned true to satisfy C_p.

The computation of PCI involves three main stages. First, for each constraint in F, the $weakening$ rule is applied to iteratively remove all non-objective literals, resulting in a simplified constraint containing only objective variables (Lines 3–4). Next, the weakened constraint is converted into a cardinality constraint using the $round2card$ rule, with its degree b representing the minimum number of literals that must be true to satisfy it (Lines 5–6). The conflict intensity of the constraint is calculated as the ratio $b/\left|Var(C_p^{card})\right|$, where $\left|Var(C_p^{card})\right|$ is the number of variables involved in the cardinality constraint (Line 7). Finally, the average of all constraint conflict intensities (i.e., dividing by the total number of constraints $|F|$) is obtained (Line 9).

Example 1 *Suppose given F including $C_1 : 1x_1 + 3x_2 + 5x_3 + 3z_1 \geq 6$, $C_2 : 1x_4 + 1x_5 \geq 1$ with objective function $O : x_1+x_2+x_3+x_4+x_5$. First, the $weakening$ rule is applied on the non-objective variable z_1, i.e., weaken(C_1, z_1), yielding $C_1' : 1x_1 + 3x_2 + 5x_3 \geq 3$. Next, $round2card$ converts C_1' into a cardinality constraint: $C_1^{card} : x_1+x_2+x_3 \geq 1$. The degree of C_1^{card} is 1, and it involves 3 objective variables, so PCI$_1 = \frac{1}{3}$. C_2 does not require any transformation, and PCI$_2 = \frac{1}{2}$. Therefore, the PCI for this instance is $\frac{5}{12}$.*

Algorithm 2: Calculate PCI

Input : PB formula F and objective function O
Output: PCI

```
1  PCI := 0;
2  foreach C_p ∈ F do
3      foreach ℓ_i ∉ Lit(O) do
4          C_p := weaken(C_p, ℓ_i);
5      C_p^card := round2card(C_p);
6      b := C_p^card.getDegree();
7      PCI_p := b/|Var(C_p^card)|;
8      PCI := PCI + PCI_p;
9  PCI := PCI/|F|;
10 return PCI;
```

The RTORC Strategy

In PBO, the assignments of objective literals directly affect solution quality. Therefore, the TORC strategy aims to assign the objective literals as close as possible to their ideal values. However, if an objective literal is frequently assigned false and repeatedly appears in conflicts, it indicates that setting this literal to false incurs a high cost and frequently triggers conflicts. To address this issue, we introduce the concept of $activity$ of objective literals, which records the conflict frequency of each objective literal and dynamically adjusts its phase assignment accordingly.

Specifically, we propose a scoring method, referred to objective literal conflict decay sum (OLCDS), which maintains an $activity$ measure for each objective literal and initially set to 0. The update rules are as follows.

- **Objective Literal Activity Bump:** When a conflict arises, for each objective literal that is assigned false within the conflicting constraint, its $activity$ is increased by an increment inc, where inc is initially set to 1.
- **Objective Literal Activity Decay:** To emphasize recent conflicts, we adopt a multiplicative decay scheme. Specifically, after every N_{confl} conflicts, multiply each literal's activity value by a factor $\rho < 1$.

The OLCDS scoring mechanism resembles the widely used variable state independent decaying sum (VSIDS) heuristic in SAT solvers (Moskewicz et al. 2001), but with two differences. First, OLCDS tracks the $activity$ of only objective literals, while VSIDS considers all variables. Second, OLCDS increments the $activity$ when an objective literal is assigned false, while VSIDS increments activity for all conflict-involved variables regardless of assignment.

By monitoring the recent conflict behavior of objective literals, we propose a new phase selection strategy called

Algorithm 3: RTORC

Input : PB formula F, objective function O, the branching variable x_b, the current best solution α^*

Output: $\alpha(x_b)$

1 **if** *PCI has not been computed* **then**
2 $\quad PCI := PCI(F, O)$;
3 $SumAct := 0$;
4 **foreach** $\ell_i \in Lit(O)$ **do**
5 $\quad SumAct := SumAct + activity[\ell_i]$;
6 $AvgAct := \frac{SumAct}{|Lit(O)|}$;
7 **if** $x_b \in Var(O)$ **then**
8 $\quad \ell_b = toLit(O, x_b)$;
9 $\quad$ **if** $PCI < T_{pci}$ && $activity[\ell_b] < AvgAct$ **then**
10 $\quad\quad \alpha(x_b) := \bar{\ell_b}$;
11 $\quad$ **else** $\alpha(x_b) := \alpha^*(x_b)$;
12 **else**
13 $\quad \alpha(x_b) := \alpha^*(x_b)$;
14 **return** $\alpha(x_b)$;

RTORC, with its pseudocode presented in Algorithm 3. The algorithm consists of two main stages. The first stage (Lines 1–6) computes the potential conflict intensity value (PCI), as well as the total and average activity of all target literals, denoted as $SumAct$ and $AvgAct$, respectively. In the second stage (Lines 7–14), $AvgAct$ is used as a threshold to determine the phase of the branching variable.

More specifically, in the phase determination stage, the algorithm first checks whether the variable x_b is an objective variable. If not, its phase is set according to the assignment in the optimal solution α^* (Line 13). If x_b is an objective variable, its corresponding literal ℓ_b is obtained using $toLit(O, x_b)$ (Line 8). If PCI is below the threshold T_{pci} and the $activity$ of ℓ_b is less than $AvgAct$, x_b is optimistically set to $\bar{\ell_b}$ (Lines 9–10). Otherwise, its phase is set to the value in α^* (Line 11).

Let $m = |F|$, $n = |Var(F)|$, and $k = max\{|C||C \in F\}$. The computational complexity of PCI is $O(mk)$, and the process of updating literal activities requires $O(n)$ time.

Compared to the original TORC strategy (Nadel 2019), the RTORC strategy introduces a refinement mechanism to the optimistic strategy. However, since this mechanism primarily relies on localized conflict statistics, it may still apply optimistic assignments to a large number of objective literals. Therefore, it is essential to further employ PCI to constrain the optimistic assignment. Notably, PCI is computed only once based on the original constraints and objective function, excluding any learned constraints. Because a single computation is sufficient to capture the structural characteristics of the instance.

A New Rephase Strategy

This section presents a reinforcement learning based rephase strategy (RLRP) that learns from solution information gathered during the search to guide phase resetting effectively.

Implementation Scheme

In the context of PBO, we regard the problem of selecting the appropriate phase for each variable as a reinforcement learning task. Since the new solution is strictly better than the previous one, we attribute the improvement to the differences in variable assignments between the two solutions. By rewarding or penalizing the corresponding assignments, the algorithm dynamically adjusts each variable's phase preference. The learned phase preferences are then used to reset the saved phase at restart, providing more targeted guidance for subsequent search.

Each variable has two possible assignments: false or true. Accordingly, we define two assignment preference probabilities, $p_f(x_i)$ and $p_t(x_i)$, which represent the probabilities of a variable x_i being assigned false or true, respectively. When a better solution is found, the algorithm updates the probabilities by comparing the assignments of each variable between the new solution and previous one. If a variable remains unchanged between the new and previous solutions, the assignment is considered correct and is given a stability reward. Otherwise, if the assignment has changed, the new assignment is regarded as a potential contributor to the improvement and is granted an exploratory reward, while the original assignment is penalized accordingly.

Assignment Preference Probability Update The timing for updating the probabilities and applying the rephase strategy is triggered whenever the restart condition is met. Initially, the assignment preference probabilities for all variables are set to 0.5. During the restart, the solver checks whether a better solution has been found. If so, it compares the variable assignments between the current best solution and the previous best solution. If a variable retains the same assignment z in both solutions, it is typically indicative that this assignment contributes to obtaining a better solution. Consequently, the assignment z is rewarded, while the probability of the opposite assignment $1 - z$ is decreased. The corresponding probability of variable x_i is then updated as follows:

$$p_j(x_i) = \begin{cases} \delta + (1-\delta) \cdot p_j(x_i), & j = z \\ (1-\delta) \cdot p_j(x_i), & j = 1 - z \end{cases} \tag{2}$$

where δ $(0 < \delta < 1)$ is a stability reward factor.

In case the assignment of a variable changes, the original assignment z is penalized, while the new assignment $1 - z$ is rewarded. The update rule for the probability is given as follows:

$$p_j(x_i) = \begin{cases} (1-\lambda)(1-\beta) \cdot p_j(x_i), & j = z \\ \lambda + (1-\lambda) \cdot \beta + (1-\lambda)(1-\beta) \cdot p_j(x_i), & j = 1 - z \end{cases} \tag{3}$$

where β $(0 < \beta < 1)$ is the penalization factor for the incorrect assignment, and λ $(0 < \lambda < 1)$ is the exploratory reward factor for the expected assignment.

This idea is inspired by the method based on learning generative models of solutions proposed by Zhou et al. (2016), which learns from existing solutions to adjust the probability of each element belonging to different groups in grouping problems. In contrast, our work focuses on PBO, where

Algorithm 4: RLRP

Input : the flag $newSol$, the previous best solution α', and the current best solution α^*
Output: rp

1 **if** $newSol = true$ **then**
2 **foreach** $x_i \in Var(F)$ **do**
3 **if** $\alpha'(x_i) = \alpha^*(x_i)$ **then**
4 update $p_f(x_i)$ and $p_t(x_i)$ according to Eq.(2);
5 **else**
6 update $p_f(x_i)$ and $p_t(x_i)$ according to Eq.(3);
7 $p_{high} := max(p_f(x_i), p_t(x_i))$;
8 $phase := p_f(x_i) \geq p_t(x_i)\ ?\ false : true$;
9 **if** $p_{high} \geq T_p$ **then** $rp[x_i] := phase$;
10 **else**
11 $p_{keep} := (T_p - p_{high})\ /\ (T_p - 0.5)$;
12 $r :=$ a random number $\in [0, 1)$;
13 **if** $r \geq p_{keep}$ **then** $rp[x_i] := phase$;
14 **else**
15 $rp[x_i] := rand()\%2 > 0\ ?\ phase : 1 - phase$;
16 **return** rp;

each variable has only a binary decision: assigned either true or false. Therefore, we adopt a modified probability update formula that learns from historical solutions to estimate each variable's tendency toward a particular assignment.

The RLRP Strategy The RLRP strategy is invoked at every restart because a restart marks a natural boundary in the search process, where the current trajectory tends to encounter frequent conflicts and struggles to achieve further improvement. Algorithm 4 shows the pseudocode of RLRP. If a new solution is found, the algorithm first updates the assignment preference probabilities for each variable according to Equations (2) and (3) (Lines 1–6). Then, based on the updated probabilities, the saved phases are reset to reflect the current preferences observed during the search (Lines 7–16). We denote the variable phase after applying the rephase strategy as rp.

Based on the updated probabilities, the preferred assignment for each variable is determined. To introduce some randomness, we propose a probability-based selection method. High assignment preference probabilities are directly set, while low probabilities are determined based on the probability values. Specifically, p_{high} and $phase$ represent the higher probability values from $p_f(x_i)$ and $p_t(x_i)$, and their corresponding assignments (Lines 7–8). If p_{high} exceeds the threshold T_p, the phase of variable x_i is set to $phase$ (Line 9). Otherwise, the algorithm calculates a retention probability p_{keep} to decide whether to retain the assignment corresponding to the higher assignment preference (Line 11). The higher the p_{high}, the greater the probability of retaining $phase$; if p_{high} is lower, there is a certain probability of choosing $1 - phase$ (Lines 12–15). The computational complexity of RLRP is $O(|Var(F)|)$.

Experiments

Experimental Preliminaries

Benchmarks. We conduct experiments on benchmarks from the PB competition 2016, 2024, and 2025. (1) `PB16`[2]: 1600 instances from the OPT-SMALLINT-LIN track. PB16 is widely used for evaluating PBO solvers, such as (Nieuwenhuis et al. 2024; Devriendt 2020; Devriendt, Gleixner, and Nordström 2021); (2) `PB24`[3]: 478 instances from the OPT-LIN track. PB24 is constructed by randomly selecting instances from many widely recognized public sources, such as MIPLIB, KNAP, and multverif, as well as various combinatorial optimization benchmarks like Vertex Cover and MaxCut. Balanced sampling across structurally similar domains is employed to ensure comprehensive and fair evaluation. (3) `PB25`[4]: 555 instances from the OPT-LIN track. The instance selection follows the method used in the PB competition 2024. Therefore, we use the three competition benchmarks as the primary evaluation datasets.

Competitors. The selected solvers for comparison are the top four standalone solvers from the OPT+UNSAT category of the OPT-LIN track in the PB Competition 2024. Among them, the first-place solver is the MIP solver SCIP (v 9.2.0)[5], while the remaining three solvers are specifically designed for PBO, including RoundingSat (v nolog)[6], RoundingSat (v log)[6], and Exact[7]. We also evaluate PBO-IHS (Smirnov, Berg, and Järvisalo 2022), an implicit hitting set-based PBO solver that does not participate in the competition.

Implementation. We implement the proposed methods on two versions of RoundingSat[6] (log and nolog) and Exact[7]. The resulting solvers are referred to as RoundingSat-PS-log, RoundingSat-PS-nolog, and Exact-PS. All experiments are conducted on Intel Xeon Gold 6238 CPU @ 2.10GHz with 512GB RAM under CentOS 7.9. Based on preliminary experiments, for the RLRP strategy, all three algorithms use the same configuration: $\delta = 0.4$, $\lambda = 0.4$, $\beta = 0.2$, and $T_p = 0.6$. For the RTORC strategy, the parameters are configured as follows: for Exact-PS, $T_{pci} = 0.1$, $N_{confl} = 9 \times 10^7$, and $\rho = 0.95$; for RoundingSat-PS-log and RoundingSat-PS-nolog, $T_{pci} = 0.15$, $N_{confl} = 9 \times 10^8$, and $\rho = 0.95$. For the LS module, all three algorithms use a maximum iteration limit of $step = 2 \times 10^8$. All compared algorithms from the PB Competition 2024 are executed with the configuration parameters provided in the competition. Each solver is run only once, with a time limit of 3600s.

Metric Setting. The evaluation metrics are divided into two aspects. The first aspect assesses the solver's capability to solve instances, i.e., the number of instances for which the solver can prove optimality or infeasibility within a given cutoff time. Specifically, #O and #U denote the number of solved optimal and unsatisfiable instances, respectively, and #S = #O + #U. The second aspect evaluates the solver's ability to obtain high-quality solutions, measured by the quality

[2]https://www.cril.univ-artois.fr/PB16/
[3]https://www.cril.univ-artois.fr/PB24/
[4]http://www.cril.univ-artois.fr/PB25/
[5]https://scipopt.org/index.php\#download
[6]https://gitlab.com/MIAOresearch/software/roundingsat
[7]https://gitlab.com/nonfiction-software/exact

Solver	PB16(#inst 1600)				PB24(#inst 478)				PB25(#inst 555)			
	#O	#U	#S	#win	#O	#U	#S	#win	#O	#U	#S	#win
SCIP	947	105	1052	1038	241	13	254	282	271	**19**	290	278
RoundingSat-nolog	915	114	1029	1006	240	13	253	287	269	16	285	335
RoundingSat-log	915	114	1029	1009	244	13	257	288	267	16	283	328
Exact	914	**117**	1031	1012	236	**14**	250	276	277	**19**	296	325
PBO-IHS	854	93	947	896	224	12	236	234	286	6	292	317
RoundingSat-PS-nolog	943	114	1057	1137	**254**	13	**267**	354	305	18	323	409
RoundingSat-PS-log	**949**	114	**1063**	1152	**254**	13	**267**	360	**308**	18	**326**	413
Exact-PS	944	116	1060	**1194**	249	**14**	263	**381**	292	**19**	311	**425**

Table 1: Experiment results on PB16, PB24, and PB25.

of objective function values. Specifically, for each instance F_j, the best solution among all solvers for the same instance F_j is denoted as $best_{F_i}$. #win denotes the number of instances where the solver finds the best solution $best_{F_i}$.

Comparison With Competition Solvers

Table 1 presents a comparison between our proposed solvers and four top-ranking solvers, as well as PBO-IHS. Overall, our solvers outperform all competitors on the PB16, PB24, and PB25 benchmarks in terms of the number of solved instances (#S) and solution quality (#win). They also achieve improvements over their respective base solvers. Specifically, on the PB16 benchmark, RoundingSat-PS-nolog, RoundingSat-PS-log, and Exact-PS solve 28, 34, and 29 more instances than their respective base versions, with improvements of 131, 143, and 182 in solution quality, respectively. On the PB24 benchmark, they solve 14, 10, and 13 more instances, with corresponding improvements of 67, 72, and 105 in solution quality. For PB25, they handle 38, 43, and 15 more instances, accompanied by improvements of 74, 85, and 100 in solution quality.

Considering runtime, we introduce the PAR-2 metric, which is calculated as the total runtime plus twice the timeout for unsolved instances. Since LS introduces a clear runtime overhead, we report the PAR-2 scores for the three proposed solvers without LS (i.e., RoundingSat-PS-nolog-noLS, RoundingSat-PS-log-noLS, and Exact-PS-noLS) to isolate the true contribution of our two main innovations, RTORC and RLRP. Specifically, on benchmarks PB16, PB24, and PB25, the algorithms RoundingSat-PS-nolog-noLS, RoundingSat-PS-log-noLS, and Exact-PS-noLS achieved average improvements of 122.11s, 140.76s, and 42.83s, respectively.

In the PB Competition 2025, the independent solver RoundingSat+pbsuma-opt-log[4] ranked first in the PBO track and employed a preprocessor `pbsuma`, designed to automatically handle symmetries in PB formulas. We incorporate `pbsuma` into our proposed solver RoundingSat-PS-log and conduct experiments on the PB25. Experimental results show that our solver outperforms RoundingSat+pbsuma-opt-log by solving 10 more instances, achieving 95 better solutions in terms of solution quality.

Effectiveness of Proposed Strategies

To analyze the effectiveness of each individual strategy in our proposed solvers, we implement three ablated variants by selectively disabling specific components: 1) noLS: the local search module is removed; 2) noRTORC: the RTORC-based phase selection strategy is disabled and replaced with the previous solution-based phase saving strategy (Demirovic and Stuckey 2019); 3) noRLRP: the RLRP strategy is removed.

Solver	PB16(#inst 1600)			
	#b(#S)	#w(#S)	#b(#win)	#w(#win)
Exact-PS-noLS	**20**	5	**546**	307
Exact-PS-noRTORC	**12**	6	**295**	227
Exact-PS-noRLRP	**7**	2	**304**	244
RoundingSat-PS-nolog-noLS	**18**	6	**412**	352
RoundingSat-PS-nolog-noRTORC	**13**	5	**292**	236
RoundingSat-PS-nolog-noRLRP	**11**	0	**301**	247
RoundingSat-PS-log-noLS	**21**	6	**368**	296
RoundingSat-PS-log-noRTORC	**9**	3	**307**	235
RoundingSat-PS-log-noRLRP	**10**	1	**289**	233
	PB24(#inst 478)			
	#b(#S)	#w(#S)	#b(#win)	#w(#win)
Exact-PS-noLS	**9**	2	**196**	64
Exact-PS-noRTORC	**8**	4	**98**	52
Exact-PS-noRLRP	**5**	1	**92**	57
RoundingSat-PS-nolog-noLS	**12**	7	**183**	104
RoundingSat-PS-nolog-noRTORC	**9**	3	**86**	51
RoundingSat-PS-nolog-noRLRP	**9**	2	**80**	59
RoundingSat-PS-log-noLS	**8**	3	**166**	99
RoundingSat-PS-log-noRTORC	**9**	4	**80**	46
RoundingSat-PS-log-noRLRP	**6**	0	**81**	52
	PB25(#inst 555)			
	#b(#S)	#w(#S)	#b(#win)	#w(#win)
Exact-PS-noLS	**13**	6	**83**	33
Exact-PS-noRTORC	**3**	0	**93**	30
Exact-PS-noRLRP	**5**	0	**90**	32
RoundingSat-PS-nolog-noLS	**19**	7	**222**	43
RoundingSat-PS-nolog-noRTORC	**5**	1	**75**	51
RoundingSat-PS-nolog-noRLRP	**3**	0	**62**	43
RoundingSat-PS-log-noLS	**17**	3	**204**	39
RoundingSat-PS-log-noRTORC	**7**	2	**65**	34
RoundingSat-PS-log-noRLRP	**3**	0	**64**	37

Table 2: Results of our proposed solvers and different versions.

The experimental results are presented in Table 2. The metrics #b(#S) and #w(#S) indicate the number of instances on which solves more or fewer problems than each ablated variant, respectively. Similarly, #b(#win) and #w(#win) reflect the number of instances for which solves obtains better or worse solutions in terms of quality. The results demonstrate that all proposed strategies contribute positively to the overall performance of solves.

Parameter Sensitivity Analysis

To analyze parameter sensitivity, we construct two evaluation sets and run experiments with Exact-PS and RoundingSat-PS-log. In the first set, we randomly select three instances from each domain (i.e., structurally similar categories) in PB16 and PB24, resulting in a total of 147 instances. The second set consists of instances whose solving statuses differ between the baseline solvers (Exact and RoundingSat-log) and their enhanced versions, totaling 61 and 71 instances, respectively. Testing parameters on these

Exact-PS									RoundingSat-PS-log							
T_{pci}	#w	ρ	#w	N_{confl}	#w	$step$	#w	T_{pci}	#w	ρ	#w	N_{confl}	#w	$step$	#w
0.05	-7	0.9	-7	9e5	-5	2e5	-13	0.05	-3	0.9	-8	9e5	-8	2e5	-11
0.1	0	0.93	-6	9e6	-4	2e6	-4	0.1	-1	0.93	-5	9e6	-6	2e6	-6
0.15	-6	**0.95**	0	5e7	-4	2e7	-9	**0.15**	0	**0.95**	0	5e7	-3	2e7	-6
0.2	-4	0.97	-5	**9e7**	0	**2e8**	0	0.2	-7	0.97	-4	9e7	-5	**2e8**	0
0.3	-9	0.99	-5	9e8	-1	1e9	-10	0.3	-11	0.99	-5	**9e8**	0	1e9	-3

Table 3: Parameter sensitivity analysis.

status-sensitive instances makes it easier to observe the impact of each parameter on solver performance. For each parameter, we evaluate five different values.

The results show that both algorithms are insensitive to the four RLRP parameters (δ, β, λ, and T_p), with fluctuations in the number of solved instances within 2 for Exact-PS and within 3 for RoundingSat-PS-log. In contrast, the three parameters in RTORC, i.e., T_{pci}, ρ, and N_{confl}, and the iteration limit $step$ of the LS module are key factors influencing solver performance. Table 4 reports the performance variations under different parameter settings, measured relative to the best-performing configuration. The column #w denotes the number of instances for which performance degrades.

Complementarity of PBO and MIP Solvers

This subsection presents experimental evidence of the complementarity between PBO complete solvers and MIP solvers. On the PB16, PB24, and PB25 benchmarks, RoundingSat-PS-nolog, RoundingSat-PS-log, and Exact-PS solve 211, 212, and 244 instances that SCIP cannot, while SCIP solves 159, 151, and 205 instances that these solvers fail to handle. We further analyze the fundamental limitations between PBO complete solvers and MIP solvers. SCIP struggles on pure SAT-structured instances because the LP relaxation provides very little information. In contrast, exact PBO solvers excel on SAT-structured instances with short constraints (especially binary and ternary constraints) due to strong Boolean propagation. However, they tend to perform poorly on instances with high-degree cardinality constraints, where propagation is harder to detect.

Based on this observation, we further propose three hybrid solvers: HyRoundingSat-PS-nolog, HyRoundingSat-PS-log, and HyExact-PS, which combine SCIP and their respective PB solvers. For comparison, we select mixed-bag[8] and UWrMaxSat-SCIP[9], which are the top-ranked hybrid solvers in the 2024 and 2025 competitions, respectively. mixed-bag is built upon RoundingSat (Elffers and Nordström 2018) and incorporates the preprocessor PaPILO (Gleixner, Gottwald, and Hoen 2023) as well as SCIP. UWrMaxSat-SCIP combines a MaxSAT algorithm UWrMaxSat (Piotrów 2020) with SCIP. To ensure a fair comparison, we allocate 300 seconds of runtime to SCIP in all three of our proposed hybrid solvers as well as in the two comparison solvers.

[8]https://bitbucket.org/coreo-group/pbcomp24-cg/src/main/

[9]https://github.com/marekpiotrow/UWrMaxSat. UWrMaxSat-SCIP is the top-ranked solver in the track of CPU ranking over all selected instances, including those not supported by some solvers.

Solver	PB16(#inst 1600)				PB24(#inst 478)				PB25(#inst 555)			
	#O	#U	#S	#win	#O	#U	#S	#win	#O	#U	#S	#win
mixed-bag	971	114	1085	985	264	13	277	275	315	19	334	325
UWrMaxSat-SCIP	**1019**	115	**1134**	1062	271	13	284	309	324	17	341	361
HyRS-PS-nolog	996	114	1110	1185	270	13	283	364	**328**	19	**347**	424
HyRS-PS-log	997	114	1111	1213	270	13	283	365	**328**	19	**347**	428
HyExact-PS	1009	**117**	1126	**1261**	**274**	**14**	**288**	**393**	323	**20**	343	**445**

Table 4: Results on mixed-bag and our hybrid solvers. For brevity, we denote HyRoundingSat-PS as HyRS-PS.

The experimental results are shown in Table 4. In PB16, UWrMaxSat-SCIP achieves the best performance in terms of the number of instances solved, while HyExact-PS excels in solution quality. In PB24, HyExact-PS outperforms others in both the number of instances solved and solution quality. In PB25, HyRoundingSat-PS-nolog and HyRoundingSat-PS-log achieve the best performance in the number of instances solved, whereas HyExact-PS leads in solution quality.

Conclusion

In this work, we propose two phase selection strategies for complete CDCL-based PBO solvers. Applying these strategies to the state-of-the-art PBO solvers RoundingSat and Exact leads to performance improvements. In addition, we integrate a local search algorithm to quickly obtain an upper bound and enable hot-starting.

PBO can be viewed as a generalization of the Maximum Satisfiability Problem (MaxSAT). In the future, we plan to explore applying the two phase selection strategies proposed in this work to MaxSAT solving, aiming to further improve solver efficiency. In addition, we will investigate the deep integration of local search and complete algorithms on PBO to enable effective information transfer.

Acknowledgments

This work is supported by National Cryptologic Science Fund of China under Grant No. 2025NCSF02046, NSFC under Grant No. 61806050, Science and Technology Development Program of Jilin Province under Grants No. 20230101060JC.

References

Berre, D. L.; Marquis, P.; Mengel, S.; and Wallon, R. 2020. On Irrelevant Literals in Pseudo-Boolean Constraint Learning. In *IJCAI 2020*, 1148–1154.

Biere, A.; Heule, M.; and van Maaren, H. 2021. Frontiers in Artificial Intelligence and Applications. In *Handbook of Satisfiability*, volume 336. IOS Press.

Buss, S.; and Nordström, J. 2021. Proof complexity and SAT solving. In *Handbook of Satisfiability*, 233–350. IOS Press.

Chen, X.; Lei, Z.; and Lu, P. 2024. Deep Cooperation of Local Search and Unit Propagation Techniques. In *CP 2024*, 6–1.

Chen, Z.; Lin, P.; Hu, H.; and Cai, S. 2024. ParLS-PBO: A Parallel Local Search Solver for Pseudo Boolean Optimization. In *CP 2024*, volume 307, 5:1–5:17.

Chu, Y.; Cai, S.; Luo, C.; Lei, Z.; and Peng, C. 2023. Towards more efficient local search for pseudo-boolean optimization. In *CP 2023*, 12–1.

Cook, S. A. 1971. The Complexity of Theorem-Proving Procedures. In *STOC*, 151–158.

Demirovic, E.; and Stuckey, P. J. 2019. Techniques Inspired by Local Search for Incomplete MaxSAT and the Linear Algorithm: Varying Resolution and Solution-Guided Search. In *CP 2019*, volume 11802, 177–194.

Devriendt, J. 2020. Watched propagation of-integer linear constraints. In *CP 2020*, 160–176.

Devriendt, J. 2024. Exact: evaluating a pseudo-Boolean solver on MaxSAT problems. *MaxSAT Evaluation 2024*, 2024: 11.

Devriendt, J.; Gleixner, A.; and Nordström, J. 2021. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1): 26–55.

Devriendt, J.; Gocht, S.; Demirovic, E.; Nordström, J.; and Stuckey, P. J. 2021. Cutting to the Core of Pseudo-Boolean Optimization: Combining Core-Guided Search with Cutting Planes Reasoning. In *AAAI 2021*, 3750–3758.

Elffers, J.; and Nordström, J. 2018. Divide and conquer: Towards faster pseudo-boolean solving. In *IJCAI*, volume 18, 1291–1299.

Gleixner, A.; Gottwald, L.; and Hoen, A. 2023. PaPILO: A parallel presolving library for integer and linear optimization with multiprecision support. *INFORMS Journal on Computing*, 35(6): 1329–1341.

Ihalainen, H.; Berg, J.; Järvisalo, M.; and Bogaerts, B. 2025. Symmetric Core Learning for Pseudo-Boolean Optimization by Implicit Hitting Sets. In *CP 2025*, volume 340, 15:1–15:26.

Jiang, L.; Ouyang, D.; Zhang, Q.; and Zhang, L. 2024. DeciLS-PBO: an effective local search method for pseudo-Boolean optimization. *Frontiers in Computer Science*, 18(2): 182326.

Le Berre, D.; and Wallon, R. 2021. On dedicated cdcl strategies for pb solvers. In *SAT 2021*, 315–331.

Lei, Z.; Cai, S.; Luo, C.; and Hoos, H. 2021. Efficient local search for pseudo boolean optimization. In *SAT 2021*, 332–348.

Lester, M. M. 2021. Scheduling reach mahjong tournaments using pseudoboolean constraints. In *SAT 2021*, 349–358.

Lester, M. M. 2022. Pseudo-Boolean optimisation for RobinX sports timetabling. *Journal of Scheduling*, 25(3): 287–299.

Lomis, O.; Devriendt, J.; Bierlee, H.; and Guns, T. 2025. Improving Reduction Techniques in Pseudo-Boolean Conflict Analysis. In *SAT 2025*, volume 341, 21:1–21:17.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *DAC 2001*, 530–535.

Nadel, A. 2019. Anytime Weighted MaxSAT with Improved Polarity Selection and Bit-Vector Optimization. In *FMCAD 2019*, 193–202.

Nieuwenhuis, R.; Oliveras, A.; Rodríguez-Carbonell, E.; and Zhao, R. 2025. Symbolic Conflict Analysis in Pseudo-Boolean Optimization. In *SAT 2025*, volume 341, 23:1–23:18.

Nieuwenhuis, R. L. M.; Oliveras Llunell, A.; Rodríguez Carbonell, E.; and Zhao, R. 2024. Speeding up pseudo-Boolean propagation. In *SAT2024*, 22:1–22:18.

Piotrów, M. 2020. UWrMaxSat: Efficient Solver for MaxSAT and Pseudo-Boolean Problems. In *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, 132–136.

Pipatsrisawat, K.; and Darwiche, A. 2007. A Lightweight Component Caching Scheme for Satisfiability Solvers. In *SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, 294–299.

Sellmann, M.; and Ansótegui, C. 2006. Disco - Novo - GoGo: Integrating Local Search and Complete Search with Restarts. In *AAAI 2006*, 1051–1056.

Smirnov, P.; Berg, J.; and Järvisalo, M. 2021. Pseudo-boolean optimization by implicit hitting sets. In *CP 2021*, 51–1.

Smirnov, P.; Berg, J.; and Järvisalo, M. 2022. Improvements to the implicit hitting set approach to pseudo-boolean optimization. In *SAT 2022*, 13–1.

Zhao, Y.; Wang, Y.; Chu, Y.; Zhou, W.; Cai, S.; and Yin, M. 2025. Improving Local Search Algorithm for Pseudo Boolean Optimization. *Journal of Artificial Intelligence Research*, 83: 1–38.

Zhou, W.; Zhao, Y.; Wang, Y.; Cai, S.; Wang, S.; Wang, X.; and Yin, M. 2023. Improving local search for pseudo boolean optimization by fragile scoring function and deep optimization. In *CP 2023*, 41–1.

Zhou, Y.; Hao, J.-K.; and Duval, B. 2016. Reinforcement learning based local search for grouping problems: A case study on graph coloring. *Expert Systems with Applications*, 64: 412–422.