Improving Local Search for Pseudo Boolean Optimization by Fragile Scoring Function and Deep Optimization

Wenbo Zhou
School of Information Science and Technology, Northeast Normal University, Changchun, China
Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China
Key Laboratory of Symbolic Computation and Knowledge Engineering of MOE, Jilin University, Changchun, China

Yujiao Zhao
School of Information Science and Technology, Northeast Normal University, Changchun, China

Yiyuan Wang[1]
School of Information Science and Technology, Northeast Normal University, Changchun, China
Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China

Shaowei Cai
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China

Shimao Wang
School of Information Science and Technology, Northeast Normal University, Changchun, China

Xinyu Wang
School of Information Science and Technology, Northeast Normal University, Changchun, China

Minghao Yin[1]
School of Information Science and Technology, Northeast Normal University, Changchun, China
Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China

Abstract

Pseudo-Boolean optimization (PBO) is usually used to model combinatorial optimization problems, especially for some real-world applications. Despite its significant importance in both theory and applications, there are few works on using local search to solve PBO. This paper develops a novel local search framework for PBO, which has three main ideas. First, we design a two-level selection strategy to evaluate all candidate variables. Second, we propose a novel deep optimization strategy to disturb some search spaces. Third, a sampling flipping method is applied to help the algorithm jump out of local optimum. Experimental results show that the proposed algorithms outperform three state-of-the-art PBO algorithms on most instances.

2012 ACM Subject Classification Computing methodologies → Search methodologies

Keywords and phrases Local Search, Pseudo-Boolean Optimization, Deep Optimization

Digital Object Identifier 10.4230/LIPIcs.CP.2023.41

Supplementary Material *Software (Source Code)*: `https://github.com/yiyuanwang1988/DeepOpt-PBO`, archived at `swh:1:dir:a1d58cf93325bed3675f5872f42adf459a518a92`

[1] corresponding author

Funding This work is supported by the National Natural Science Foundation of China under Grant No.61806050, the Natural Science Research Foundation of Jilin Province of China under Grant Nos.YDZJ202201ZYTS412 and YDZJ202201ZYTS423, Ministry of Education under Grant No.2021BCF01002, CCF-Huawei Populus Grove Fund, and the Fundamental Research Funds for the Central Universities under Grant Nos.2412023YQ003, 2412022ZD015, 2412022ZD018, 2412022QD040 and 93K172022K10.

1 Introduction

The Boolean Satisfiability (SAT) problem is a prototypical NP-complete problem, whose aim is to determine whether a given propositional formula is satisfiable or not. The SAT problem plays a core role in many domains of computer science and artificial intelligence [14]. Many real-world problems can be encoded into SAT and its optimization version MaxSAT and solved using their powerful solvers. However, due to the limited expressive power of SAT and MaxSAT, their encodings often generate very large problem instances. For such cases, pseudo-Boolean optimization (PBO) provides a more expressive and natural way to express constraints than SAT and MaxSAT [23]. Besides, PBO is very close enough to SAT to benefit from the recent advances in SAT solving [23].

The PBO consists of a set of pseudo-Boolean constraints and an objective function, whose goal is to find a solution that minimizes the objective function and satisfies all pseudo-Boolean constraints. Solvers for PBO can be divided into complete and incomplete solvers. Up to now, many complete PBO solvers have been proposed, such as Sat4j [3], Open-WBO [21], NaPS [24], RoundingSat [12, 10], RSCard [13], RS/lp [9] and PBO-IHS [26, 27].

Compared to complete PBO solvers, there are relatively fewer incomplete solvers on PBO. The reason may be that PBO is more complicated than SAT and how to find a suitable variable to flip is still difficult. As one of the most popular incomplete approaches, local search can find an approximate solution within a reasonable time [32, 33]. Lei et al. [18] proposed a novel local search algorithm called LS-PBO to handle PBO. Key features of LS-PBO include a converting method to obtain corresponding objective constraints, a weighting scheme to guide the search direction, and a well-designed scoring function to flip some candidate variables. Besides, for some special cases of PBO such as NK-Landscapes and MAX-kSAT, a new perturbation strategy called VIGbP was proposed [29]. According to the literature, the current best incomplete algorithm for PBO is LS-PBO.

In this work, to further improve the performance of local search algorithms on solving PBO, we propose a local search framework for PBO based on three main ideas.

First, we present a two-level selection strategy to choose which variable to flip. In our proposed algorithm, we use the previous scoring function *score* [18] as a primary scoring function, which is defined as the decrease of the total penalty of related constraints and objective function. To address the issue about tie-breaking in the primary scoring function, we propose a fragile scoring function *hhscore* as the secondary scoring function. The proposed *hhscore* can greatly differentiate between two kinds of satisfied constraints by using the definition of satisfied threshold.

Second, we propose a novel deep optimization strategy (DeepOpt) to deeply probe some regions using locked and unlocked operations during the local search. Recently, the DeepOpt strategy was first proposed by Chen et al. [8] and has been successfully applied in solving dominating set problems. In our proposed DeepOpt strategy, we preferentially select some variables in unsatisfied constraints as unlocked operations. Moreover, we use some trigger conditions to decide whether the algorithm calls DeepOpt or not.

Third, we design a sampling flipping method to modify a current candidate solution when the algorithm tramps into local optimum. In the proposed method, we adopt two sampling ways to collect some samples among all unsatisfied constraints. We further employ a probabilistic heuristic "best from multiple selections" (BMS) [6] to select a candidate variable. Besides, we also apply a new scoring function to simultaneously flip two variables.

By incorporating these three ideas and some other tricks, we develop two local search algorithms for PBO. Extensive experiments are carried out to evaluate our algorithms on the benchmarks used in the literature. Experimental results show that the proposed algorithms outperform three state-of-the-art PBO algorithms on almost all the benchmarks.

2 Preliminaries

Since a non-linear pseudo-Boolean (PB) constraint can be translated into an equivalent set of linear PB constraints [23], we only start with a review of the basics of linear PB constraints here. A linear PB constraint is defined over a finite set of Boolean variables. Boolean variable x_i can take only two values *false* (0) and *true* (1). A literal l_i over a Boolean variable x_i is either x_i or $\overline{x}_i = 1 - x_i$. A linear PB constraint is a 0-1 integer inequality.

$$\sum_i a_i l_i \rhd b \tag{1}$$

where a_i and b are integer constants, l_i are literals and $\rhd \in \{=, >, \geq, <, \leq\}$ is one of the classical relational operators. All PB constraints can be normalized into the following form.

$$\sum_i a_i l_i \geq b \tag{2}$$

where all the literals l_i are distinct, and all the coefficients a_i and the *degree* b are non-negative integers.

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 $O : \sum_i c_i l_i$ where c_i is a non-negative integer coefficient. Given a PB constraint $C_p : \sum_i a_i^p l_i^p \geq b^p$, the sum of its coefficients is defined as $sum(C_p) = \sum_i a_i^p$, its average coefficient is defined as $coeff(C_p) = sum(C_p)/|L(C_p)|$ where $L(C_p)$ is the set of literals in C_p, and its maximum coefficient a_{max}^p is the maximum value of coefficients in C_p. The average coefficient of an objective function O is defined as $coeff(O) = \sum_i c_i/|L(O)|$ where $L(O)$ is the set of literals in O.

Given a PB formula F, its complete assignment is a mapping that assigns 0 or 1 to each variable. Given a complete assignment of F, if a literal evaluates to true, we say it is a *true literal* and otherwise it is a *false literal.* A PB constraint C_p is *satisfied* when the left and right terms of the constraint evaluate to integers which satisfy the relational operator. Otherwise, C_p is *unsatisfied.* The sum of coefficients of true literals in C_p is denoted as $SatL(C_p) = \sum_{l_i^p = 1 \wedge l_i^p \in C_p} a_i^p$. An assignment α of F is feasible if and only if α satisfies all PB constraints in F. The value of the objective function of a feasible solution α is denoted as $obj(\alpha)$. The PBO problem aims to obtain a feasible solution for F with the minimum objective value.

2.1 Review for Weighting and Scoring Function

Constraint weighting techniques have usually been used to guide and diversify the search process [31, 5, 15]. The weighting based scoring function *score* for PBO is recently proposed by Lei et al. [18]. Each PB constraint $C_p \in F$ and an objective function O have the property of weighting, denoted as $w(C_p)$ and $w(O)$, respectively. Before introducing the weighting, we first present a basic concept. Given a PB formula F and m is the number of constraints, the average constraint coefficient of F is defined as $avg\_coeff = \sum_{p=1}^{m} w(C_p) \times coeff(C_p)/m$.

The property of weighting works as follows.

Weighting Rule 1: At first, $w(O) = 1$ and $w(C_p) = 1$ for an objective function O and each PB constraint C_p.

Weighting Rule 2: For each unsatisfied constraint C_p, $w(C_p) = w(C_p) + 1$.

Weighting Rule 3: If the current $obj(\alpha)$ of the objective function is better than the current best-found value of objective function during the search process so far and $w(O) \times coeff(O) - avg\_coeff \leq \zeta$, then $w(O) = w(O) + 1$. In our work, we use the same parameter value of ζ as [18], i.e., $\zeta = 100$.

Given an assignment α of F, if a PB constraint $C_p : \sum_i a_i^p l_i^p \geq b^p$ is unsatisfied, the *penalty* of C_p is defined as $w(C_p) \times (b^p - \sum_i a_i^p l_i^p)$. For the objective function $O : \sum_i c_i l_i$, the *penalty* of O is defined as $w(O) \times \sum_i c_i l_i$. Based on the definition of *penalty*, we introduce two scoring functions including hard score *hscore* and objective score *oscore* as below. For a Boolean variable x_i, the respective $hscore(x_i)$ and $oscore(x_i)$ are the decrease of the total penalty of unsatisfied PB constraints and the objective function caused by flipping x_i. Combing the above scoring functions, we define the score of a variable x_i as $score(x_i) = hscore(x_i) + oscore(x_i)$. Remark that after flipping some variables during the search process, the corresponding *score* values should be updated accordingly.

3 Two-Level Selection Strategy

In this section, we introduce a secondary scoring function to reinforce local search algorithms for PBO and then propose a two-level selection strategy to decide a candidate variable.

3.1 Fragile Scoring Function

As a core guidance for the search process, scoring functions play an important role in local search algorithms, which measure the benefits of a candidate variable. In local search algorithms for PBO, such benefits *hscore* can be set as the distance between the sum of coefficients of true literals and the corresponding degree, whereas *oscore* can be set as the distance between the current and best-found objective values [18].

In our algorithm, we consider the sum of *hscore* and *oscore* (i.e., *score*) as the primary scoring function, which is the same method as LS-PBO [18]. According to our preliminary experiments, 9% candidate variables on average have the same largest *score* value during the search. To address the issue about tie-breaking in the primary scoring function, previous work uses the *age* information of variables as the secondary scoring function, where *age* is defined as the number of steps since the last time it is flipped. But the experimental results show that the use of only *age* cannot effectively guide the search process. Thus, to further choose a variable among these variables with the same best *score* value, we design a novel fragile scoring function denoted as *hhscore*.

Before introducing *hhscore*, we first introduce a necessary concept. For a PB constraint C_p, $gap(C_p) = min\{b^p + a_{max}^p, sum(C_p)\}$ is used to denote the *satisfied threshold* of C_p, which plays a key role in our proposed *hhscore*. Based on the satisfied threshold of PB constraints, we define a fragile satisfied PB constraint in the following.

▶ **Definition 1.** *For a satisfied PB constraint C_p (i.e., $SatL(C_p) \geq b^p$), the C_p is a fragile satisfied PB constraint if and only if $SatL(C_p) < gap(C_p)$.*

On the one hand, if a satisfied PB constraint C_p is fragile, we think that flipping any variable in this PB constraint would probably make this constraint become unsatisfied. On the other hand, if a satisfied PB constraint is not fragile (i.e., $SatL(C_p) \geq gap(C_p)$), we think this PB constraint is solid, which means that flipping any true literal in C_p would not make C_p become unsatisfied with a high probability. Although we have also tried a similar property that measures the number of true literals in a satisfied PB constraint C_p, such as $gap(C_p) = b^p + 1$, we did not find it useful in our algorithm.

To maintain the information of "fragile" for each literal during the search process, we define *inner* as below.

▶ **Definition 2.** *Suppose that a Boolean variable x_i whose literal l_i appears in some PB constraints (e.g., C_p) and the coefficient of l_i in C_p is a_i^p. The value of $inner(x_i, C_p)$ is calculated as follows.*

(a) *C_p is an unsatisfied PB constraint, i.e., $SatL(C_p) < b^p$:*
- *If l_i is a true literal in C_p, $inner(x_i, C_p) = 0$;*
- *If l_i is a false literal in C_p, $inner(x_i, C_p) = max\{a_i^p - (b^p - SatL(C_p)), 0\}$.*

(b) *C_p is a fragile satisfied PB constraint, i.e., $b^p \leq SatL(C_p) < gap(C_p)$:*
- *If l_i is a true literal in C_p, $inner(x_i, C_p) = -min\{a_i^p, SatL(C_p) - b^p\}$;*
- *If l_i is a false literal in C_p, $inner(x_i, C_p) = min\{a_i^p, gap(C_p) - SatL(C_p)\}$.*

(c) *C_p is satisfied but not a fragile PB constraint, i.e., $gap(C_p) \leq SatL(C_p)$:*
- *If l_i is a true literal in C_p, $inner(x_i, C_p) = -max\{a_i^p - (SatL(C_p) - gap(C_p)), 0\}$;*
- *If l_i is a false literal in C_p, $inner(x_i, C_p) = 0$.*

The value of $inner(x_i, C_p)$ is subject to three distinct factors including the state of C_p (i.e., *satisfied* or *unsatisfied*), the value of l_i (i.e., *true* or *false*), and the coefficient of l_i (i.e., a_i^p). To make the readers easily understand the above Definition 2, we list all the situations corresponding to different *inner* values in Figure 1.

Considering all the PB constraints in which variable x's literal appears, the proposed fragile scoring function *hhscore* of variable x_i is defined as below.

$$hhscore(x_i) = \sum_{p=1}^{n_x} inner(x_i, C_p) \tag{3}$$

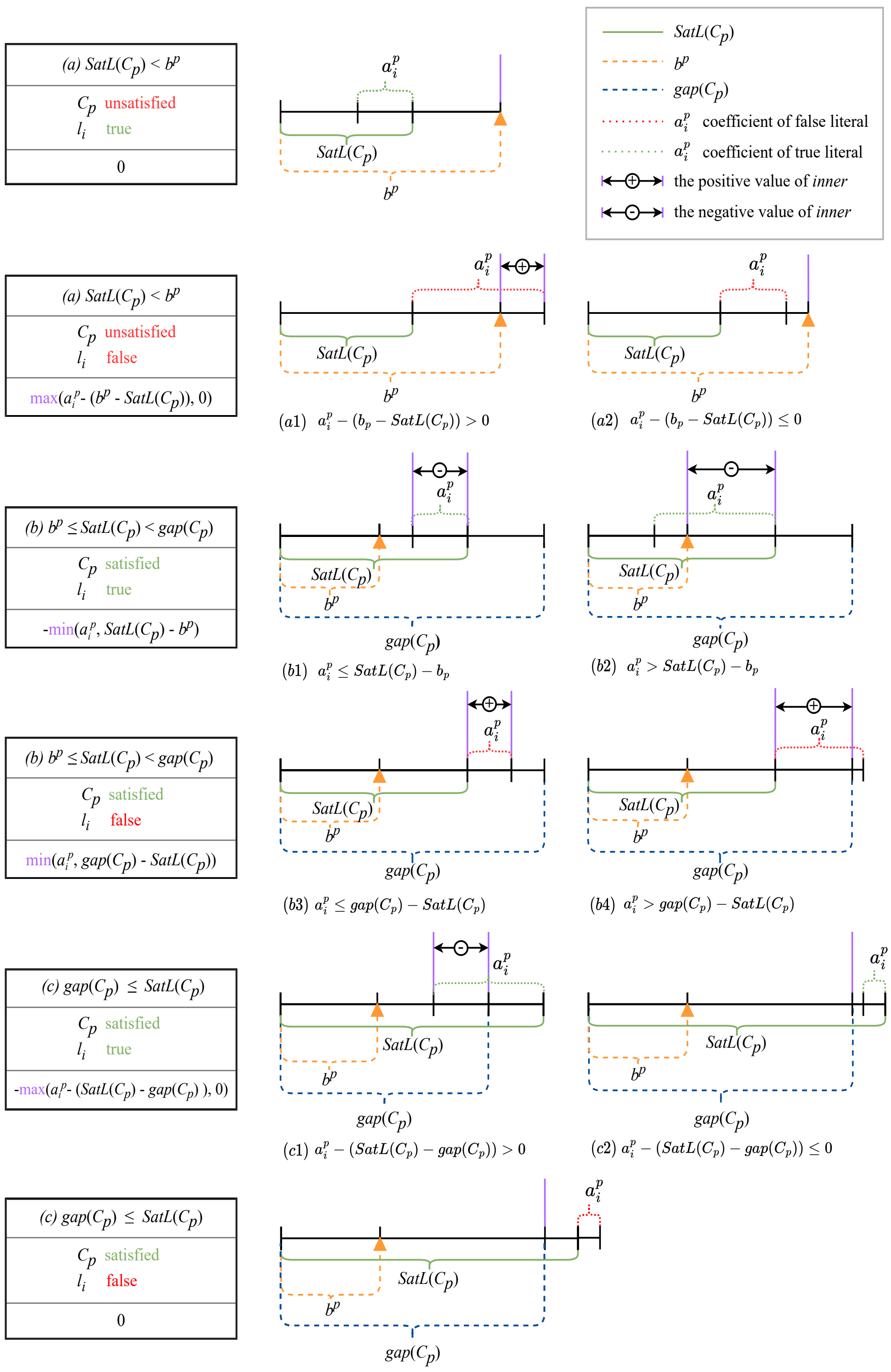
where n_x is the number of PB constraints including the literal of x_i.

3.2 Selection Rule

Combining the respective advantages of *score* and *hhscore*, we propose a two-level selection strategy as follows.

Flipping Rule. Flip a variable x_i with the biggest $score(x_i)$ value, breaking ties by preferring the one with the biggest $hhcore(x_i)$ value, further ties are broken randomly.

The proposed secondary scoring function is inspired by *subscore* [7], but has two essential differences. First, our proposed scoring function can be considered as a general version of *subscore* because the value of satisfied threshold *gap* is equal to 2 for the SAT problem, which is the same function as *subscore*. Second, previous work uses a linear combination of *subscore* and another scoring function as the primary scoring function, whereas our work considers a two-level scoring function to guide the search process. In addition, experiments show that the trigger fraction of using *hhscore* at least once for all tested instances is about 99.6%.



(a) SatL(Cp) < b^p
Cp unsatisfied
li true
0
SatL(Cp)
b^p
gap(Cp)
a_i^p coefficient of false literal
a_i^p coefficient of true literal
the positive value of inner
the negative value of inner
(a) SatL(Cp) < b^p
Cp unsatisfied
li false
max(a_i^p - (b^p - SatL(Cp)), 0)
(a1) a_i^p − (b_p − SatL(C_p)) > 0
(a2) a_i^p − (b_p − SatL(C_p)) ≤ 0
(b) b^p ≤ SatL(Cp) < gap(Cp)
Cp satisfied
li true
-min(a_i^p, SatL(Cp) - b^p)
(b1) a_i^p ≤ SatL(C_p) − b_p
(b2) a_i^p > SatL(C_p) − b_p
(b) b^p ≤ SatL(Cp) < gap(Cp)
Cp satisfied
li false
min(a_i^p, gap(Cp) - SatL(Cp))
(b3) a_i^p ≤ gap(C_p) − SatL(C_p)
(b4) a_i^p > gap(C_p) − SatL(C_p)
(c) gap(Cp) ≤ SatL(Cp)
Cp satisfied
li true
-max(a_i^p - (SatL(Cp) - gap(Cp)), 0)
(c1) a_i^p − (SatL(C_p) − gap(C_p)) > 0
(c2) a_i^p − (SatL(C_p) − gap(C_p)) ≤ 0
(c) gap(Cp) ≤ SatL(Cp)
Cp satisfied
li false
0

Figure 1 A graphical explanation of *inner*.

Algorithm 1 DeepOpt.

```
Input: PBO instance $F$ and a perturbation initialization assignment $\alpha$
Output: A perturbation solution $\alpha$ of $F$
1  $UnSatSet := unsat(\alpha)$ and $SatSet := F \setminus UnsatSet$;
2  $CandSet := \emptyset$;
3  while $|CandSet| \leq \gamma \times n$ do
4      if $UnSatSet \neq \emptyset$ then
5          select a random unsatisfied PB constraint $C$;
6          $UnSatSet := UnSatSet \setminus \{C\}$;
7          for each variable $x \in C$ do
8              if $x$'s literal is true && with 50% probability then $\alpha := \alpha$ with $x$ flipped ;
9              $CandSet := CandSet \cup \{x\}$;
10     else
11         select a random satisfied PB constraint $C$;
12         $SatSet := SatSet \setminus \{C\}$;
13         for each variable $x \in C$ do
14             $CandSet := CandSet \cup \{x\}$;
15     if $|unsat(\alpha)| > MaxHard$ then break ;
16 for $step = 0; step < L_{opt}; step{+}{+}$ do
17     select a variable $x$ among $|CandSet|/2$ samples from $CandSet$ based on Flipping Rule;
18     $\alpha := \alpha$ with $x$ flipped;
19 return $\alpha$;
```

4 Deep Optimization for PBO

Local search algorithms usually search the entire space and focus on exploring some promising spaces using several heuristic strategies. Recently, a general perturbation mechanism called deep optimization has been proposed by Chen et al. [8], which can deeply probe some regions based on locked and unlocked operations and then converge to a new solution quickly. Note that deep optimization is somewhat similar to some classic search frameworks such as large neighborhood search [25]. According to this general framework, we propose a new deep optimization approach DeepOpt for PBO.

The pseudo-code of our proposed DeepOpt is reported in Algorithm 1 and the corresponding trigger conditions of DeepOpt will be displayed in the next section. We use the $unsat(\alpha)$ function to denote the set of unsatisfied PB constraints under an assignment α. At first, the algorithm uses *UnSatSet* and *SatSet* to store unsatisfied and satisfied PB constraints under a perturbation initialization assignment, respectively (Line 1). Afterward, a candidate variable set *CandSet* is initialized to an empty set (Line 2), which stores all unlocked variables in the following search. Note that, in our work, we use *CandSet* to store all unlocked variables which means that these variables can be flipped in the following search phase, while the remaining variables can be seen as locked variables.

The DeepOpt usually consists of two phases including a selection phase (Lines 3–15) to generate several candidate local search spaces and a search phase (Lines 16–18) to repair these search spaces.

In the first phase, we flip several variables to achieve the purpose of early preparation. There are two exit conditions in this phase. The first one is that $|CandSet|$ is larger than $\gamma \times n$ where n is the number of variables (Line 3), whereas the second one is that the number of unsatisfied constraints under the current assignment is larger than *MaxHard* (Line 15) where γ and $MaxHard$ are two parameters in DeepOpt. At each iteration, if $UnSatSet$ is not empty, the algorithm preferentially chooses a random unsatisfied PB constraint C from $UnSatSet$ (Lines 4–5) and puts all variables of C into $CandSet$ (Line 9). For each variable $x \in C$, if x's literal in C is true, it occurs with 50% probability to flip x (Line 8). If the algorithm selects a random satisfied constraint C, the algorithm only adds all variables of C into $CandSet$ (Line 14). Note that according to our preliminary experiments, if the first phase only selects the candidate variables and does not flip these variables, then the performance of DeepOpt will become bad.

In the second phase, the algorithm applies the search process to perturb the assignment α until the limit of iterations L_{opt} is reached (Line 16). The algorithm employs the BMS strategy [6] in the process of selecting a candidate variable, i.e., randomly choosing $|CandSet|/2$ variables to compose a candidate set. Afterward, the algorithm flips a variable x among a candidate set based on the flipping rule (Lines 17–18). At last, the algorithm returns the final assignment α (Line 19).

5 DeepOpt-PBO Algorithm

In this section, we propose an effective local search algorithm DeepOpt-PBO for PBO, whose main framework is presented in Algorithm 2. The DeepOpt-PBO is mainly divided into the initialization and search phases.

In the initialization phase (Lines 1–3), the best solution α^* is set to $\emptyset$ and its objective value obj^* is set to $+\infty$. To obtain an initial assignment α, all variables are set to 0. The algorithm initializes the related weight information according to the weighting rule 1.

In the following, there is an outer loop (Lines 4–32) and an inner loop (Lines 7–31). During the search, whenever a better feasible assignment is obtained, α^* and obj^* are updated accordingly (Line 12). After each inner loop, the algorithm uses the $RestartVar$ function to restart a current assignment (Line 32), which will be presented in Section 5.2. Finally, the algorithm returns α^* and obj^* when reaching a time limit.

In each inner loop ($step < L$), the algorithm searches for a local optimal assignment α. The algorithm uses $GoodSet$ to store variables whose $score$ is larger than 0 (Line 15). If $GoodSet$ is not empty, the algorithm flips a candidate variable based on the proposed flipping rule (Lines 16–18). Otherwise, it means that the algorithm tramps into local optimum. The algorithm uses the modified weighting rule 2 and weighting rule 3 to update the corresponding weight value (Lines 20–21). Specifically, we optimize previous weighting rule 2, resulting in a novel modified weighting rule 2, which will be mentioned in Section 5.2. Moreover, the algorithm will use a sampling technique $SampleFlip$ to flip one or more variables to help the algorithm itself jump out of local optimum, which will be introduced in the next subsection (Line 22).

Trigger Conditions of DeepOpt. In the below part, we will introduce some trigger conditions of DeepOpt in our proposed algorithm. At first, three variables are defined as below.

1) $unhard$ is used to denote the number of unsatisfied PB constraints. Before each inner loop, $unhard$ is initialized to the number of unsatisfied constraints under the current assignment (i.e., $|unsat(\alpha)|$) (Line 6). In the inner loop, whenever $|unsat(\alpha)| < unhard$, $unhard$ is updated accordingly (Lines 8–9). After calling the DeepOpt method, $unhard$ will be updated by $|unsat(\alpha)|$ (Line 31).

Algorithm 2 DeepOpt-PBO.

```
Input  : PBO instance $F$ and cutoff time $cutoff$
Output : An assignment $\alpha^*$ of $F$ and its objective value
1  $\alpha^* := \emptyset$ and $obj^* := +\infty$;
2  $\alpha :=$ all variables are set to 0;
3  initialize the weight value of the objective function and each constraint to 1;
4  while $elapsed\ time < cutoff$ do
5      $step_{opt} := 1$ and $coff := 1$;
6      $unhard := |unsat(\alpha)|$;
7      for $step = 1; step < L; step{+}{+}$ do
8          if $|unsat(\alpha)| < unhard$ then
9              $unhard := |unsat(\alpha)|$;
10             $step_{opt} := step_{opt}/2$ and $step := step/2$;
11         if $\alpha$ is feasible && $obj(\alpha) < obj^*$ then
12             $\alpha^* := \alpha$ and $obj^* := obj(\alpha)$;
13             $step_{opt} := step := 1$;
14             if $coff \neq 1$ then $coff := coff/2$;
15         $GoodSet := \{x \mid score(x) > 0\}$;
16         if $GoodSet \neq \emptyset$ then
17             select a variable $x$ in $GoodSet$ based on Flipping Rule;
18             $\alpha := \alpha$ with $x$ flipped;
19         else
20             update the weight of each constraint based on Modified Weighting Rule 2;
21             update the weight of the objective function based on Weighting Rule 3;
22             $SampleFlip(F, \alpha)$;
23         $step_{opt} := step_{opt} + 1$;
24         if $step_{opt} \% (coff \times MinStep) == 0$ then
25             if $|unsat(\alpha)| \leq MinHard$ && with 50% probability then
26                 if $coff \neq \delta$ then $coff := coff \times 2$;
27                 $DeepOpt(F, \alpha)$;
28             else if $\alpha^* \neq \emptyset$ then
29                 $\alpha := \alpha^*$;
30                 $DeepOpt(F, \alpha)$;
31             $unhard := |unsat(\alpha)|$ and $step_{opt} := 1$;
32     $RestartVar(F, \alpha)$;
33 return $(\alpha^*, obj^*)$;
```

2) $step_{opt}$ records the non-improvement steps after the initialization phase or the last DeepOpt operation. Before each inner loop, the algorithm sets $step_{opt}$ to 1 (Line 5). In each iteration of the inner loop, $step_{opt}$ is increased by 1 (Line 23). When the algorithm obtains a better assignment (Line 13) or the algorithm calls the DeepOpt method (Line 27 or 30), $step_{opt}$ will be set to 1 (Line 31). If $|unsat(\alpha)| < unhard$, then $step_{opt}$ will be cut in half (Lines 8 and 10).

3) *coff* is used to control the frequency of using the DeepOpt method. Before each inner loop, *coff* is initialized to 1 (Line 5). When the algorithm finds a better assignment, the value of *coff* is divided by 2 to reduce the frequency of perturbations (Line 14). If $|unsat(\alpha)|$ is smaller than parameter *MinHard*, it means unsatisfied PB constraints are few enough to consider perturbations, avoiding tramping into a local optimum. The value of *coff* will be doubled with a 50% probability (Line 26). During this process, parameter δ controls the maximum value of *coff*.

The algorithm judges whether to call the DeepOpt method under each ($coff \times MinStep$) iteration (Line 24). If $|unsat(\alpha)| \leq MinHard$, the algorithm calls the DeepOpt method with a 50% probability (Line 27). Otherwise, if α^* is not empty, the algorithm will use α^* as a perturbation initialization assignment and then employ the DeepOpt method (Lines 28–30).

Algorithm 3 SampleFlip.

```
   Input  : PBO instance $F$ and an assignment $\alpha$
   Output : A modified assignment $\alpha$
1  if $|unsat(\alpha)| == 0$ then
2      select a random variable $x$ with $oscore(x) > 0$;
3      $\alpha := \alpha$ with $x$ flipped;
4      return $\alpha$;
5  select a random unsatisfied constraint $C$;
6  $CSet := VSet := \emptyset$;
7  if $|unsat(\alpha)| \geq \beta$ then
8      for $i = 1$ to $\beta$ do
9          select a random unsatisfied constraint $C_i$;
10         $CSet := CSet \cup \{C_i\}$;
11 if $|L(C)| == 1$ then
12     select only variable $x$ in $C$ and $\alpha := \alpha$ with $x$ flipped;
13 else if $|L(C)| == 2$ then
       /* Two variables $x_1$ and $x_2$ in $C$                                */
14     select a variable $x_i$ with the largest $score_t(x_1, x_i)$ value, breaking ties randomly;
15     select a variable $x_j$ with the largest $score_t(x_2, x_j)$ value, breaking ties randomly;
16     if $score_t(x_1, x_i) > 0$ || $score_t(x_2, x_j) > 0$ then
17         if $score_t(x_1, x_i) > score_t(x_2, x_j)$ then
18             $\alpha := \alpha$ with $x_1$ and $x_i$ flipped;
19         else $\alpha := \alpha$ with $x_2$ and $x_j$ flipped;
20     else
21         select a variable $x$ among $x_i$ and $x_j$ based on Flipping Rule and $\alpha := \alpha$ with $x$
             flipped;
22 else
23     if $|unsat(\alpha)| \geq \beta$ then
24         for each constraint $C_p \in CSet$ do
25             select $|L(C_p)|/2$ samples from $L(C_p)$ and put them into $VSet$;
26     else
27         select $|L(C)|/2$ samples from $L(C)$ and put them into $VSet$;
28     select a variable $x$ from $VSet$ based on Flipping Rule;
29     $\alpha := \alpha$ with $x$ flipped;
30 return $\alpha$;
```

5.1 Sampling Flipping

The pseudo-code of *SampleFlip* is outlined in Algorithm 3. First, we introduce a new scoring function, which is used in our *SampleFlip* method. When flipping two variables x_s and x_z simultaneously, $score_t(x_s, x_z)$ is defined as the sum of the decrease of the total penalty of unsatisfied PB constraints and the objective function. Although it is easy to see the computation complexity of *score* is quite lower than $score_t$, $score_t$ can find a better flipping operation compared to *score*. The method of selecting more candidate elements simultaneously has also been used in some different NP-hard problems [28, 34]. Moreover, in the *SampleFlip* method, we only consider $score_t$ in a special case that the number of literals in the selected constraint C is 2 (i.e., $|L(C)| = 2$).

In the beginning, if there are no unsatisfied constraints, the algorithm selects and flips a random variable x with $oscore(x) > 0$ (Lines 1–4). There are two main important components, including clause sampling (Lines 5–10) and variable flipping (Lines 11–29) in the algorithm. In the phase of clause sampling, the algorithm adopts two kinds of sampling ways. The first sampling method is to select a random unsatisfied constraint C (Line 5). If $|unsat(\alpha)| > \beta$, then the algorithm activates the second sampling method, i.e., adding β unsatisfied constraints into $CSet$ (Lines 7–10).

In the phase of variable flipping, if C is a unit clause (i.e., $|L(C)| = 1$), the algorithm flips the only variable x in C (Lines 11–12). If C is a binary clause (i.e., $L(C) = \{x_1, x_2\}$), the algorithm tries to find two pairs of variables $\{x_1, x_i\}$ and $\{x_2, x_j\}$ with the largest $score_t$ value (Lines 14–15). If there exists a positive flipping operation among these two pairs, the algorithm flips a pair with the better $score_t$ value (Lines 16–19). Otherwise, the algorithm still flips only one variable among x_1 and x_2 based on the flipping rule (Lines 20–21). In the subsequent process, the algorithm will depend on the value of parameter β to collect some samples from $CSet$ or C into $VSet$ (Lines 23–27). At last, the algorithm selects and then flips the best variable x from $VSet$ (Lines 28–29).

The intuitive explanations behind *SampleFlip* come from two aspects. First, single flipping mechanism for local search is easy to fall into a local optimum, while sampling strategy can explore the solution space more effectively in a look-ahead way. Second, the sampling strategy selects variables from various unsatisfied constraints, providing more search directions. In addition, we specially focus on the case of two literals, to keep running time low but a good performance.

5.2 Some Other Techniques

Based on the main part as shown above, we also introduce some additional heuristics to further improve the efficiency, including a weighting rule and a restart strategy.

Modified Weighting Rule 2. The weighting scheme plays an important role in the search process. By increasing the weight of unsatisfied constraints, we can accurately guide the search process toward more efficient directions. In each iteration of the inner loop, if *GoodSet* is empty, the visited times of each unsatisfied constraint will be increased by 1. For an unsatisfied constraint C_p, when its visited times are larger than $b^p/coeff(C_p)$, the value of $w(C_p)$ will be increased by 1, and the value of its visited times is reset to 0.

Restart Strategy. The second adopted technique is the restart strategy. Under a current assignment α, we can change the original value of each variable with a certain probability, i.e., from 1 (0) to 0 (1). Besides, the restart strategy will reset the weight value of each constraint according to the weighting rule 1. If the number of literals in an objective function O is larger than β (i.e., $|L(O)| > \beta$), then $w(O)$ will be reset based on the weighting rule 1, too.

6 Experimental Evaluation

We first introduce the seven selected benchmarks, three state-of-the-art PBO competitors, and the adopted experimental setup. Then, we carry out extensive experiments to evaluate the performance of our proposed algorithm.

6.1 Experiment Preliminaries

Since the literature on local search algorithms for handling PBO is very sparse, we selected all used instances from [18, 10]. To be specific, we considered 3738 instances obtained from three application benchmarks and four standard benchmarks: (1) 24 instances from the minimum-width confidence band problem (MWCB) [2]. These MWCB instances were obtained based on the MIT-BIH arrhythmia database[2]. (2) 18 instances from the wireless sensor network optimization problem (WSNO) [16, 17]. We used the same encoding as previous work [18] to obtain an optimization version of WSNO. (3) 21 instances from the seating arrangements problem (SAP) [1]. These instances were originally proposed in the MaxSAT Evaluation 2017. (4) 1600 OPT-SMALL-INT instances from the most recent PB Competition in 2016 (PB2016)[3]. The PB2016 benchmark is often considered as the main target for comparing against some other PB solvers [10, 27]. (5) The 0-1 integer linear programming optimization benchmark (MIPLIB) contains 267 instances from the mixed integer programming library MIPLIB 2017[4]. (6) 1025 crafted combinatorial instances (CRAFT) are provided in the literature [30]. (7) The Knapsack benchmark (KNAP) consists of a total of 783 instances [22].

Table 1 Tuned parameters of our proposed algorithms.

Parameter	Range	Final value
DeepOpt		
MinStep	$\{10^3,10^4,10^5,10^6\}$	10^5
δ	{64,128,256}	128
γ	{0.02,0.05,0.08,0.11}	0.05
MaxHard	{30,50,70,90}	50
MinHard	{5,10,15}	10
L_{opt}	{10,30,50,70}	50
Some other parameters in our proposed algorithms		
β	{50,100,150,200}	100
L	$\{10^2,10^3,10^4,10^5\}$	10^4

According to the frequency of using the proposed restart strategy, we propose two versions of DeepOpt-PBO, resulting in DeepOpt-PBO-v1 and DeepOpt-PBO-v2. In detail, DeepOpt-PBO-v1 does not use the restart strategy (i.e., parameter L is set to INT_MAX). DeepOpt-PBO-v2 is run for half of the computation time given to the algorithm with the restart strategy, while the second half of the available computation time of DeepOpt-PBO-v2 is given to the algorithm without the restart strategy. We also attempted to use the restart strategy during the whole time, and the result is not satisfactory. According to our preliminary experiments by using the automatic configuration tool irace [20], Table 1 shows the parameter values. Specifically, since these benchmarks have different scales, we built a training set and randomly selected 10 instances from the corresponding tested benchmark. The tuning process is given a budget of 5000 runs for the training set with a time budget of 3600s per run.

The proposed algorithms are compared against three state-of-the-art PBO algorithms, including a local search algorithm LS-PBO [18] and two exact PBO solvers, i.e., RoundingSat [10] and PBO-IHS [27]. Although mixed integer programming solvers (e.g., SCIP [4]) and

[2] http://physionet.org/physiobank/database/mitdb/
[3] http://www.cril.univ-artois.fr/PB16/
[4] http://miplib.zib.de

MaxSAT solvers (e.g., CASHWMaxSAT-CorePlus [19]) can be directly applied to solving PBO, we mainly focus on evaluating the performance of specialized solvers for PBO. The codes of all competitors were kindly provided by the authors. As for LS-PBO, RoundingSat and PBO-IHS, we employ the default parameters in the corresponding literature, respectively. Our code will be made publicly available. All algorithms are implemented in C++ and compiled by g++ with -O3 option. All the algorithms are run on Intel Xeon Gold 6238 CPU @ 2.10GHz with 512GB RAM under CentOS 7.9.

For the application benchmarks, our proposed algorithms and LS-PBO are both run 20 times whose seed is from 1 to 20 on each instance, whereas the exact solvers are run once on each instance. For the other four standard benchmarks (i.e., PB2016, MIPLIB, CRAFT and KNAP), following the settings of previous works [10, 27], all the algorithms are run only once on each instance. We test the algorithms with a time limit of 3600 seconds.

For the application benchmarks, we use *min* to denote the best solution value found and *avg* to denote the average value over the 20 runs. For all the benchmarks, we report the number of instances where the algorithm finds the best solution value among all algorithms, denoted by *#win*. There are some unsatisfied instances in the PB2016 benchmark. RoundingSat and PBO-IHS can guarantee the optimality of the solutions they obtain and thus can prove some of these unsatisfied instances, whereas DeepOpt-PBO and LS-PBO cannot do it because these two algorithms belong to incomplete algorithms. For the above case, following the similar method from the literature [18], if all the algorithms fail to obtain any feasible solution for such an unsatisfied instance, then *#win* value of all the algorithms for this instance needs to be increased by 1. The bold value indicates the best solution value obtained by all the algorithms. In detail, the bold value of each *avg* column indicates the best average solution when some algorithms obtain the same minimal solution values. For one instance, if only one algorithm finds the best minimal solution value, only its corresponding *min* column should be marked.

6.2 Experimental Results

Note that for the application benchmarks, two exact solvers RoundingSat and PBO-IHS can obtain the same best solution as our proposed algorithms for only 4 instances. Thus, for the sake of space, we do not report the detailed results of these two exact solvers. We mainly compare DeepOpt-PBO-v1 and DeepOpt-PBO-v2 with LS-PBO. The experimental results on the application benchmarks are presented in Tables 2–4. According to our results, our proposed algorithms are consistently superior on the MWCB and SAP benchmarks. For the WSNO benchmark, two proposed algorithms and LS-PBO can obtain the same best solution values. Furthermore, LS-PBO can find a minimal average solution for 15 instances, while our proposed algorithms do it for only 6 instances. Our proposed algorithms adopt some perturbation mechanisms, which may make them difficult for the algorithms to steadily obtain a good solution.

Table 5 gives a summary on all the benchmarks. According to the experimental results, DeepOpt-PBO-v2 outperforms other algorithms in terms of obtaining more optimal solution values on PB2016, MIPLIB and CRAFT, except for the benchmark KNAP where PBO-IHS has a better performance. In addition, under a given time limit, we observe that PBO-IHS can prove the optimality of a solution for more instances compared to RoundingSat in all benchmarks. Although our proposed algorithms cannot prove the optimality due to the natural property of local search, DeepOpt-PBO-v2 can find more minimal solution values overall. Because some instances from the above benchmarks have different kinds of problem structures and all the local search algorithms are run only once on each instance, the restart

Table 2 Experiment results on MWCB.

Instance	DeepOpt-PBO-v1		DeepOpt-PBO-v2		LS-PBO	
	min	avg	min	avg	min	avg
1000_200_90	**103363**	104188.35	103430	104233	110450	111314.25
1000_250_90	**140223**	141182.2	140237	141194.9	148181	149828.65
1200_200_90	**104063**	**104572.25**	**104063**	104589.1	110993	112897.1
1200_250_90	**141211**	142310.6	141238	142343.2	150212	152888.7
1400_200_90	**103948**	104867.35	104057	104901.55	110792	112975.65
1400_250_90	**141567**	142675.7	141746	142722.15	150981	152932.9
1600_200_90	**119226**	**120182.4**	**119226**	120215.9	136944	143371.9
1600_250_90	**162651**	163893.3	162656	163937.1	183797	196547.75
1800_200_90	**203135**	**205888.75**	**203135**	205959.2	219536	224144.95
1800_250_90	**253073**	257501.75	253078	257715.25	276336	283192.95
2000_200_90	**227294**	229591.95	227424	229676.1	246109	250958.4
2000_250_90	**286970**	**289889.4**	**286970**	290058.1	309645	314528.5
1000_200_95	**113384**	114749.8	113501	114855.95	117064	117945.7
1000_250_95	**151882**	153581.05	152007	153622	156543	157976.1
1200_200_95	**114585**	115920.2	114675	115975	118045	119544.4
1200_250_95	**153104**	155765.8	153138	155798.35	159310	161454.1
1400_200_95	**114195**	**115231.3**	**114195**	115305.95	118913	119779.4
1400_250_95	**155158**	156149.65	155174	156180.15	161658	162917.95
1600_200_95	**168271**	**171985.9**	**168271**	172024.6	185707	190763.55
1600_250_95	**215266**	218013.5	215316	218069.55	236547	244060.35
1800_200_95	**238699**	241690.55	238702	241779.85	251744	256988.75
1800_250_95	**297981**	**302487.3**	**297981**	302927.25	314968	318961.25
2000_200_95	**257696**	**262062.9**	**257696**	262113.6	272832	277406.7
2000_250_95	**324379**	**328952.6**	**324379**	329032.9	340859	346007.95

Table 3 Experiment results on WSNO.

Instance	DeepOpt-PBO-v1		DeepOpt-PBO-v2		LS-PBO	
	min	avg	min	avg	min	avg
100_40_4	**210**	**210**	**210**	**210**	**210**	**210**
150_60_4	**602**	612.85	**602**	**602**	**602**	602.2
200_80_4	**715**	719.45	**715**	716.65	**715**	**715.1**
250_100_4	**1305**	1401.2	**1305**	1330.05	**1305**	**1305**
300_120_4	**1257**	1330.25	**1257**	1373	**1257**	**1257.05**
350_140_4	**1737**	1957.4	**1737**	1997.95	**1737**	**1744.05**
400_160_4	**2240**	2509.55	2241	2598.85	**2240**	**2240.5**
450_180_4	**1869**	2780.7	1878	2598.7	**1869**	**1889.25**
500_200_4	**2577**	3676.8	2674	3637.95	**2577**	**2616.2**
100_40_6	**140**	**140**	**140**	**140**	**140**	**140**
150_60_6	**402**	**402**	**402**	**402**	**402**	**402**
200_80_6	**477**	480	**477**	**477.05**	**477**	477.7
250_100_6	**870**	893.35	**870**	**870.1**	**870**	870.5
300_120_6	**839**	866.55	**839**	862.15	**839**	**839.3**
350_140_6	**1158**	1267.7	**1158**	1288.25	**1158**	**1158.85**
400_160_6	**1493**	1671.8	**1493**	1656.5	**1493**	**1494.25**
450_180_6	**1246**	1588.45	1265	1641.6	**1246**	**1247.8**
500_200_6	**1718**	1984.05	**1718**	1927	**1718**	**1727.65**

strategy plays a key role in the performance of DeepOpt-PBO-v2. To sum up, for three application benchmarks, PB2016, MIPLIB, and CRAFT, our algorithm totally dominates all the competitors, whereas PBO-IHS performs better than other PBO solvers for the KNAP benchmark.

Additionally, we evaluate the performance of all the solvers on seven benchmarks using performance profile [11]. As shown in Figure 2, the plot captures the probability of reaching a fixed quality in a time, at most a factor τ slower than the optimal algorithm. Specially,

Table 4 Experiment results on SAP.

Instance	DeepOpt-PBO-v1		DeepOpt-PBO-v2		LS-PBO	
	min	avg	min	avg	min	avg
100	**579**	**579.7**	**579**	579.8	580	583
110	**618**	**618.75**	**618**	618.85	619	626.05
120	**675**	**677.15**	**675**	677.8	679	685.6
130	**733**	736.9	734	738.4	738	744.6
140	**750**	**751.9**	**750**	753.75	757	763.65
150	**803**	**808.4**	**803**	810.3	821	828.45
160	**849**	**854.85**	**849**	855.75	867	872.75
170	**880**	**884.4**	**880**	885.55	897	907.5
180	**939**	948.45	941	951.7	971	977.4
190	**965**	973.2	970	977	996	1005.25
200	**1029**	**1040**	**1029**	1042.55	1067	1073.4
210	**1067**	1074.7	1068	1077.9	1094	1112.75
220	**1114**	**1127.3**	**1114**	1129.6	1151	1163.1
230	**1151**	1166.85	1162	1169.7	1195	1205.75
240	**1179**	1192.6	1183	1195.5	1219	1234.55
250	**1235**	1243.1	1237	1245.5	1274	1290.55
260	**1275**	1286.15	1277	1287.45	1318	1334.9
270	**1344**	1353.1	1348	1356.1	1392	1403.55
280	**1348**	1370.75	1354	1375.35	1407	1424.3
290	**1403**	1416.35	1405	1419.5	1448	1471.15
300	**1471**	1491.5	1480	1496.1	1538	1548.4

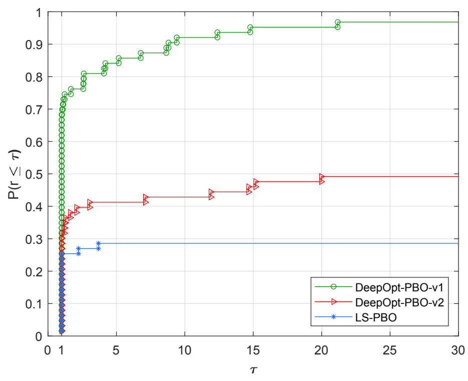
Table 5 Summary results of comparing our proposed algorithms to its competitors on all the benchmarks. #inst denotes the number of instances in each benchmark.

Benchmark	#inst	DeepOpt-PBO-v1	DeepOpt-PBO-v2	LS-PBO	RoundingSat	PBO-IHS
		#win	#win	#win	#win	#win
MWCB	24	**24**	9	0	0	0
WSNO	18	**18**	14	**18**	4	4
SAP	21	**21**	9	0	0	0
PB2016	1600	1141	**1226**	1060	1195	1101
MIPLIB	267	163	**168**	135	125	130
CRAFT	1025	903	**930**	878	882	917
KNAP	783	695	693	649	392	**778**
Total	3738	2965	**3049**	2740	2598	2930

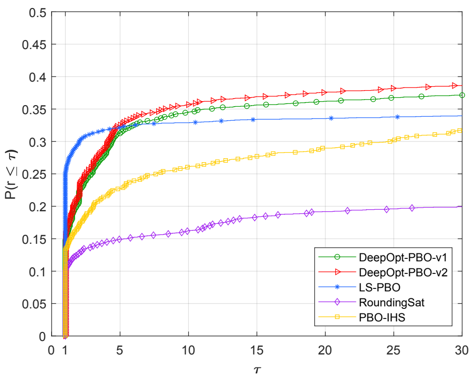
Table 6 Comparative results for DeepOpt-PBO-v1 and its modified versions with different strategies on the application benchmarks. #better and #worse denote the number of instances where DeepOpt-PBO-v1 obtains better and worse solution values, respectively.

Benchmark	#inst	vs. v1+nohh		vs. v1+nodo		vs. v1+nosf	
		#better	#worse	#better	#worse	#better	#worse
MWCB	24	**11**	8	**24**	0	**24**	0
WSNO	18	**1**	0	**1**	0	0	0
SAP	21	**11**	5	**18**	0	**20**	0
Total	63	**23**	13	**43**	0	**44**	0

when τ equals 1, we obtain the probability that the algorithm is the fastest. Figure 2 (a) demonstrates the superior performance of DeepOpt-PBO-v1 on the first three application benchmarks. It can be observed that DeepOpt-PBO-v1 has a higher probability of finding the optimal solution for each τ. In Figure 2 (b), LS-PBO performs the best within a short time scale. However, DeepOpt-PBO-v2 surpasses LS-PBO around $\tau = 6$ and maintains its leadership consistently.



(a) Three application benchmarks.



(b) Four standard benchmarks.

Figure 2 Performance profiles for DeepOpt-PBO and all competitors for reaching the best solution on the three application benchmarks (a) and four standard benchmarks (b).

For three application benchmarks, to determine better solution values, we increase the execution time to 5400 seconds, and once again our algorithms perform better than LS-PBO. For the PB2016 benchmark, when increasing the execution time to 5400 seconds, although the solution values obtained by RoundingSat and PBO-HIS are better than before, our algorithm can still find more best solution values than RoundingSat and PBO-IHS for 29 and 140 instances, respectively.

To verify the effectiveness of the proposed strategies, we compare DeepOpt-PBO-v1 with three alternative versions: 1) v1+nohh utilizes the *age* strategy instead of *hhscore*; 2) v1+nodo does not use the DeepOpt method; 3) v1+nosf does not employ the *SampleFlip* function. The results in Table 6 demonstrate that all proposed strategies are effective. In addition, we randomly select 20 instances for each standard benchmark. In total, 80 instances are picked. We also test the performance of our proposed algorithm and three alternative versions on these picked instances. All the algorithms are run 20 times on each instance. Once again, the results show that our proposed algorithm obviously performs better than three alternative versions.

Here, we give a discussion to compare our algorithm with MIP solvers and MaxSAT solvers. First, although MaxSAT solvers (e.g., CASHWMaxSAT-CorePlus [19]) can be directly applied to solving PBO, Lei et al. [18] observed that several of the instances from some benchmarks, such as PB2016, are too large to admit practical encodings into MaxSAT. Thus, MaxSAT solvers have usually poor performance for PBO. Second, more general solvers (e.g., MIP solvers) could also be used for comparison, but in our work, we mainly focus on evaluating the performance of specialized solvers for PBO. We have also tested the performance of the non-commercial MIP solver SCIP [4] on all the seven benchmarks. In detail, for the KNAP benchmark, the performance of PBO-IHS and SCIP is better than our algorithm. But, for the remaining six benchmarks, our algorithms perform better than SCIP.

7 Conclusion

In this paper, we propose a two-level selection strategy, a novel deep optimization strategy, and a sampling flipping method. Based on the above strategies and some other tricks, we develop two local search algorithms. Experiments show that the proposed algorithms significantly outperform the state-of-the-art PBO algorithms.

References

1 Carlos Ansotegui, Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT evaluation 2017: Solver and benchmark descriptions, 2017.

2 Jeremias Berg, Emilia Oikarinen, Matti Järvisalo, and Kai Puolamäki. Minimum-width confidence bands via constraint optimization. In *Proceedings of the Twenty-Third Principles and Practice of Constraint Programming*, volume 10416, pages 443–459, 2017.

3 Daniel Le Berre and Romain Wallon. On dedicated CDCL strategies for PB solvers. In *Proceedings of the Twenty-Fourth Theory and Applications of Satisfiability Testing*, volume 12831, pages 315–331, 2021.

4 Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The SCIP optimization suite 8.0. *arXiv:2112.08872*, 2021.

5 Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the Sixteenth Eureopean Conference on Artificial Intelligence*, volume 16, pages 146–150, 2004.

6 Shaowei Cai, Jinkun Lin, and Chuan Luo. Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess. *Journal of Artificial Intelligence Research*, 59:463–494, 2017.

7 Shaowei Cai and Kaile Su. Comprehensive score: Towards efficient local search for SAT with long clauses. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, pages 489–495, 2013.

8 Jiejiang Chen, Shaowei Cai, Yiyuan Wang, Wenhao Xu, Jia Ji, and Minghao Yin. Improved local search for the minimum weight dominating set problem in massive graphs by using a deep optimization mechanism. *Artificial Intelligence*, 314:103819, 2023.

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

10 Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter J. Stuckey. Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, volume 35, pages 3750–3758, 2021.

11 Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

12 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 1291–1299, 2018.

13 Jan Elffers and Jakob Nordström. A cardinal improvement to pseudo-Boolean solving. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, volume 34, pages 1495–1503, 2020.

14 John Franco and John Martin. A history of satisfiability. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 3–74. 2009.

15 Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. The MIT press, 2009.

16 Gergely Kovásznai, Balázs Erdélyi, and Csaba Biró. Investigations of graph properties in terms of wireless sensor network optimization. In *Proceedings of the IEEE International Conference on Future IoT Technologies*, pages 1–8, 2018.

17 Gergely Kovásznai, Krisztián Gajdár, and Laura Kovács. Portfolio SAT and SMT solving of cardinality constraints in sensor network optimization. In *Proceedings of the Twenty-First International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 85–91, 2019.

18 Zhendong Lei, Shaowei Cai, Chuan Luo, and Holger H. Hoos. Efficient local search for pseudo Boolean optimization. In *Proceedings of the Twenty-Fourth Theory and Applications of Satisfiability Testing*, volume 12831, pages 332–348, 2021.

19 Zhendong Lei, Yiyuan Wang, Shiwei Pan, Shaowei Cai, and Minghao Yin. CASHWMaxSAT-CorePlus: Solver description. *MaxSAT Evaluation*, page 8, 2022.

20 Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

21 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-WBO: A modular maxsat solver. In *Proceedings of the Seventeenth Theory and Applications of Satisfiability Testing*, volume 8561, pages 438–445, 2014.

22 David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9):2271–2284, 2005.

23 Olivier Roussel and Vasco Manquinho. Pseudo-Boolean and cardinality constraints. In *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1087–1129. 2021.

24 Masahiko Sakai and Hidetomo Nabeshima. Construction of an ROBDD for a PB-constraint in band form and related techniques for PB-solvers. *IEICE Transactions on Information and Systems*, E98.D(6):1121–1127, 2015.

25 Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the Fourth Principles and Practice of Constraint Programming*, volume 1520, pages 417–431, 1998.

26 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-Boolean optimization by implicit hitting sets. In *Proceedings of the Twenty-Seventh Principles and Practice of Constraint Programming*, volume 210, pages 51:1–51:20, 2021.

27 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Improvements to the implicit hitting set approach to pseudo-Boolean optimization. In *Proceedings of the Twenty-Fifth International Conference on Theory and Applications of Satisfiability Testing*, volume 236, pages 13:1–13:18, 2022.

28 Zhouxing Su, Qingyun Zhang, Zhipeng Lü, Chu-Min Li, Weibo Lin, and Fuda Ma. Weighting-based variable neighborhood search for optimal camera placement. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, volume 35, pages 12400–12408, 2021.

29 Renato Tinós, Michal W. Przewozniczek, and Darrell Whitley. Iterated local search with perturbation based on variables interaction for pseudo-Boolean optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 296–304, 2022.

30 Marc Vinyals, Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, and Jakob Nordström. In between resolution and cutting planes: A study of proof systems for pseudo-Boolean SAT solving. In *Proceedings of the Twenty-First Theory and Applications of Satisfiability Testing*, volume 10929, pages 292–310, 2018.

31 Chris Voudouris and Edward Tsang. Partial constraint satisfaction problems and guided local search. In *Proceedings of the Practical Application of Constraint Technology*, pages 337–356, 1996.

32 Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. Sccwalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artificial Intelligence*, 280:103230, 2020.

33 Yiyuan Wang, Dantong Ouyang, Liming Zhang, and Minghao Yin. A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity. *Science China Information Sciences*, 60:062103, 2017.

34 Jiongzhi Zheng, Jianrong Zhou, and Kun He. Farsighted probabilistic sampling based local search for (weighted) partial maxsat. *CoRR*, abs/2108.09988, 2021.