



# Towards More Efficient Local Search for Pseudo-Boolean Optimization

Yi Chu  

Institute of Software, Chinese Academy of Sciences, Beijing, China

Shaowei Cai<sup>1</sup>  

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Chuan Luo  

School of Software, Beihang University, Beijing, China

Zhendong Lei  

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Cong Peng  

Finovation in CCBFT, Beijing, China

---

## Abstract

Pseudo-Boolean (PB) constraints are highly expressive, and many combinatorial optimization problems can be modeled using pseudo-Boolean optimization (PBO). It is recognized that stochastic local search (SLS) is a powerful paradigm for solving combinatorial optimization problems, but the development of SLS for solving PBO is still in its infancy. In this paper, we develop an effective SLS algorithm for solving PBO, dubbed *NuPBO*, which introduces a novel scoring function for PB constraints and a new weighting scheme. We conduct experiments on a broad range of six public benchmarks, including three real-world benchmarks, a benchmark from PB competition, an integer linear programming optimization benchmark, and a crafted combinatorial benchmark, to compare *NuPBO* against five state-of-the-art competitors, including a recently-proposed SLS PBO solver *LS-PBO*, two complete PB solvers *PBO-IHS* and *RoundingSat*, and two mixed integer programming (MIP) solvers *Gurobi* and *SCIP*. *NuPBO* has been exhibited to perform best on these three real-world benchmarks. On the other three benchmarks, *NuPBO* shows competitive performance compared to state-of-the-art competitors, and it significantly outperforms *LS-PBO*, indicating that *NuPBO* greatly advances the state of the art in SLS for solving PBO.

**2012 ACM Subject Classification** Theory of computation → Randomized local search

**Keywords and phrases** Pseudo-Boolean Optimization, Stochastic Local Search, Scoring Function, Weighting Scheme

**Digital Object Identifier** 10.4230/LIPIcs.CP.2023.12

**Supplementary Material** *Software (Source Code)*: <https://github.com/filyouzicha/NuPBO>

**Funding** This work is supported by the Strategic Priority Research Program of the Chinese Academy of Sciences, Grant No. XDA0320000 and XDA0320300, the National Natural Science Foundation of China under Grant 62302492, and Grant 62202025, the Research Program of CCBFT, Grant No. KT2100040, and CCF-Huawei Populus Grove Fund under Grant CCF-HuaweiSY20231.

---

<sup>1</sup> Corresponding author.



## 1 Introduction

Recent progress in the theory and application of the Boolean satisfiability (SAT) and maximum satisfiability (MaxSAT) problems has led to the development of high-performance complete solvers [11, 23, 1, 32, 2, 4, 31] and stochastic local search (SLS) solvers [8, 28, 7, 6, 27, 25, 5, 45]. SAT and MaxSAT solvers can address challenging problems in a wide variety of fields, and they are usually designed to deal with formulas encoded in conjunctive normal form (CNF).

For problems involving cardinality constraints, CNF solvers usually become ineffective, since expressing such constraints in CNF would dramatically increase the size of the formula and introduce many auxiliary variables and clauses [30]. Linear pseudo-Boolean (PB) constraints provide a more natural and direct way to express cardinality constraints than CNF. Meanwhile, linear PB constraints stay close to CNF and can benefit from advancements in SAT solving [33]. In practice, PB constraints occur in many areas, including VLSI design, economics, computer vision, and manufacturing [43, 44, 33]. The pseudo-Boolean optimization (PBO) problem is to find a satisfying assignment to a set of PB constraints that minimizes a given objective function.

### 1.1 Related Work

Existing pseudo-Boolean solvers are primarily based on complete methods. A number of PB solvers are based on resolution: they express the PB constraints in CNF and then call conflict-driven clause learning (CDCL) solvers, such as *MINISAT+* [13], *Open-WBO* [29], and *NaPS* [34]; alternatively, they deal with the PB constraints but derive new information only in the form of clauses [17]. CDCL is somewhat limited in its reasoning in that it is based on a resolution-proof system, for which exponential lower bounds are known for simple combinatorial principles [20, 15]. Another method requires going beyond resolution and using cutting planes, which can be found in recent PB solvers such as *Sat4j* [24], *RoundingSat* [14, 12] and *RoundingSat-Card* [15]. The success of the implicit hitting set (IHS) method in MaxSAT motivates another work, i.e., the implementation of the *PBO-IHS* solver for solving PBO [36, 37]. In addition, since PB constraints can be considered as 0-1 linear constraints, mixed integer programming (MIP) solvers can be directly applied to solving PBO. Representative and high-performance MIP solvers include *SCIP* [16] - one of the fastest non-commercial solvers, and *Gurobi* [19] - one of the most powerful commercial solvers.

Stochastic local search (SLS) is recognized to be one of the most powerful techniques for solving computationally hard problems in many areas of computer science, operations research, and engineering, and it has shown great success in solving SAT and MaxSAT [22]. In the book [21], a model for local search to solve constraint problems is presented. Somewhat surprisingly, there are only a few research works on using SLS for solving PBO [3, 39, 26].

Since the introduction of dynamic local search (DLS) methods [35, 9, 40, 41], weighting schemes play critical roles in the development of high-performance SLS algorithms. Modern SLS solvers for MaxSAT employ a scoring function defined as the weighted cost of unsatisfied clauses and incorporate a clause weighting scheme to adjust the weights during the search. In particular, most SLS solvers for MaxSAT focus on improving the scoring function through carefully designed weighting schemes. In recent years, the introduction of new weighting schemes has led to breakthroughs in the SLS algorithms for the (weighted) partial MaxSAT ((W)PMS) problem. The newly proposed weighting scheme, Weighting-PMS, in the SATLike [25] algorithm significantly improves the performance of SLS for (W)PMS. Currently, the state-of-the-art SLS algorithms for (W)PMS all employ the Weighting-PMS technique [5, 45].

It is intuitive that the scoring function commonly used in SLS algorithms for MaxSAT (i.e., measuring the total weight of all unsatisfied clauses) does not work well for the PBO problem, because it does not take into account the unsatisfied degree of PB constraints. Indeed, this issue has already attracted attention in the literature, and a scoring function that considers the unsatisfied degree of PB constraints is proposed in [42] and can effectively handle linear PB-constrained problems.

A recent SLS solver called *LS-PBO* [26] has been proposed for PBO and currently represents the state of the art in SLS for PBO. For *LS-PBO*, its scoring function measures the sum of the product between the degree of violation of all unsatisfied constraints and the weights of the constraints. However, such scoring function does not consider the balance between the degree of violation among different constraints. In addition, weighting schemes have not been applied to PBO until the introduction of *LS-PBO*. The weighting scheme in *LS-PBO* resembles the existing one named Weighting-PMS, which aims to increase the weights of unsatisfied constraints and the objective function when the algorithm falls into a local optimum, and to set an upper bound on the weight of the objective function. However, in the context of PBO solving, the research of designing weighting schemes is still in its infancy, which urgently calls for more powerful weighting schemes for PBO.

## 1.2 Contributions

In this work, we focus on improving the performance of SLS for solving PBO. In particular, we propose two main ideas. The first idea is a novel scoring function that considers the violation degree of unsatisfied constraints and utilizes a smooth function to balance the violation degree of different constraints. For each constraint, its smooth function is instantiated as the average of the coefficients of all variables appearing in the corresponding constraint. Since our scoring function is equipped with a weighting scheme, our second idea is a novel weighting scheme for PBO. Rather than setting an upper bound on the weight of the objective function, we adopt a weighting scheme with a stricter condition for updating the weight of the objective function.

On the basis of these two ideas, we develop a new SLS algorithm, named *NuPBO*. We conduct experiments on 6 benchmarks, which include 3 benchmarks encoded from real-world applications, and 3 standard benchmarks. On these 6 benchmarks, *NuPBO* is compared to 5 solvers, including *LS-PBO* [26], *PBO-IHS* [37], *RoundingSat* [12], *Gurobi* [19], and *SCIP* [16]. On the 3 application benchmarks, *NuPBO* achieves improvement over *LS-PBO*, and significantly outperforms other competitors. On the other 3 benchmarks, *NuPBO* exhibits competitive performance compared to its competitors, including the commercial solver *Gurobi*. This represents a significant advance in the research of SLS solvers for PBO. In addition, we evaluate the effectiveness of the underlying ideas on all benchmarks.

## 2 Preliminaries

Given a set of  $n$  Boolean variables  $V = \{x_1, x_2, \dots, x_n\}$ , a literal is either a variable  $x_i$  or its negation  $\neg x_i$ . A clause is a disjunction of literals, i.e.,  $c_i = l_{i_1} \vee l_{i_2} \vee \dots \vee l_{i_k}$ , where  $k$  denotes the length of clause  $c_i$ . A CNF formula  $F$  is a conjunction of clauses. An assignment is a mapping that assigns a Boolean value (*True* (i.e., 1) or *False* (i.e., 0)) to each variable. Given an assignment  $\alpha$ , a clause  $c$  is satisfied if at least one literal in  $c$  is *True*; otherwise,  $c$  is unsatisfied. Given a CNF formula  $F$ , the Boolean satisfiability (SAT) problem is to decide whether an assignment exists such that all clauses are satisfied, and the maximum satisfiability (MaxSAT) problem is to find an assignment that maximizes the number of satisfied clauses.

## 12:4 Towards More Efficient Local Search for Pseudo-Boolean Optimization

A linear pseudo-Boolean constraint (LPB constraint, PB constraint for short) has the following form:

$$\sum_{j=1}^n a_j l_j \triangleright b, \quad a_j, b \in \mathbb{Z} \quad (1)$$

where  $b$  is called the degree of the constraint,  $l_j$  is a literal,  $a_j$  is the coefficient of  $l_j$ ,  $n$  is the length of the constraint,  $\triangleright$  is one of the classical relational operators ( $=, >, \geq, < \text{ or } \leq$ ), and  $\mathbb{Z}$  is the integer set.

For each Boolean variable,  $x_i = 1 - \neg x_i$ . It is important to note that for an equality constraint, there need to be two normalized constraints to represent it. Therefore, all PB constraints can be normalized into the following form:

$$\sum_{j=1}^n a_j l_j \geq b, \quad a_j, b \in \mathbb{N}_0^+ \quad (2)$$

where  $\mathbb{N}_0^+$  is the non-negative integer set [33].

In the following sections, we assume that the PB constraints are of the normalized form. A PB formula  $F$  is a conjunction of PB constraints. An assignment is a mapping that assigns a Boolean value to each variable. Given an assignment  $\alpha$ , a PB constraint  $c$  is satisfied if the corresponding inequality holds under  $\alpha$ ; otherwise,  $c$  is unsatisfied. If an assignment  $\alpha$  satisfies all constraints in  $F$ , then we say  $\alpha$  is a feasible solution (or solution for short).

A pseudo-Boolean optimization (PBO) instance consists of a PB formula  $F$  and a linear Boolean objective function  $\sum_{j=1}^n e_j l_j + d$ ,  $e_j \in \mathbb{N}^+, d \in \mathbb{Z}$ , and the task is to find an assignment that satisfies all PB constraints in  $F$  and minimizes the objective function. Given an assignment  $\alpha$ , we use  $obj(\alpha)$  to denote the value of the objective function. Given a solution  $\alpha$ , the cost of the solution  $\alpha$  is equal to  $obj(\alpha)$ . We say a solution  $\alpha_1$  is better than another solution  $\alpha_2$ , if  $obj(\alpha_1) < obj(\alpha_2)$ .

The average coefficient of a PB constraint  $c$  is denoted as  $avg_{coe}(c) = (\sum_{j=1}^n a_j)/n$ . The average coefficient of an objective function  $o$  is denoted as  $avg_{coe}(o) = (\sum_{j=1}^n e_j)/n$ . Because PB constraints must be satisfied in a PBO problem, the PB constraints are referred to as *hard constraints*.

Given an assignment  $\alpha$ , we define the value of violation of a hard constraint  $c$  as

$$viol(c) = \max \left( 0, b - \sum_{j=1}^n a_j l_j \right)$$

In other words, if the hard constraint  $c$  is satisfied under  $\alpha$ , then  $viol(c) = 0$ ; otherwise (i.e.,  $c$  is unsatisfied),  $viol(c)$  is the integer distance of  $c$  from being satisfied. In a PBO instance, a solution is an assignment, under which all hard constraints are satisfied.

The SLS algorithms that employ constraint weighting schemes usually maintain weight for each constraint. We use  $w(c)$  to represent the weight of each hard constraint  $c$ , and  $w(o)$  to represent the weight of the objective function  $o$ .

### 3 Main Ideas

In general, the search directions of SLS algorithms are guided by the scoring function. It is recognized that the effectiveness of the scoring function could be enhanced through working with a weighting scheme. In this section, we first propose a new scoring function, and then design a new weighting scheme to work with it.

■ **Table 1** The violation and objective value under all assignments of the PBO instance  $I_1$  in Example 3.1.

$viol/obj$	Assignment $(x_1, x_2, x_3)$							
	(0, 0, 0)	(0, 0, 1)	(0, 1, 0)	(0, 1, 1)	(1, 0, 0)	(1, 0, 1)	(1, 1, 0)	(1, 1, 1)
$viol(c_1)$	0	0	0	0	4	3	3	2
$viol(c_2)$	1	0	0	0	4	2	3	1
$viol(c_3)$	29	13	14	0	0	0	0	0
$obj(\alpha)$	1	1	0	0	2	2	1	1

### 3.1 A New Scoring Function

Given a PBO instance, which consists of  $n$  PB constraints (hard constraints) and one objective function, we assume that the current assignment is  $\alpha$ .

**Scoring Function in *LS-PBO*.** Before introducing our new scoring function, we first describe the existing scoring function proposed in *LS-PBO* [26], which is presented as follows.

- For a hard constraint  $c$ , the penalty function is defined as  $penalty(c) = w(c) \times viol(c)$ ; then the hard score of a variable  $x$  is defined as the decrement of the sum of the penalty function values of all hard constraints caused by flipping  $x$ , which is denoted by  $hscore(x)$ .
- For the objective function  $o$ , the value of the objective function is  $obj(\alpha)$ , and the penalty function is defined as  $penalty(o) = w(o) \times obj(\alpha)$ ; then the soft score of a variable  $x$  is defined as the decrement of the penalty function value of the objective function caused by flipping  $x$ , which is denoted by  $sscore(x)$ .
- The score of a variable  $x$  is defined as  $score(x) = hscore(x) + sscore(x)$ .

**An Intuitive View.** The above scoring function has a drawback, which is due to its underlying penalty function. The penalty function above considers the weights and  $viol$  (resp.  $obj$ ) values of hard (resp. soft) constraints. In this way, it measures the importance of a variable in a constraint  $c$  by the coefficient of the corresponding variable in  $c$ . Nevertheless, it should be noted that, as the value of  $score(x)$  is determined by all hard constraints and the objective function involving  $x$ , the above penalty function may overemphasize the importance of  $x$  in constraints with relatively large coefficients, resulting in an unreasonable value of its  $score$ . In the following, we illustrate our intuition through a simple PBO instance.

► **Example 3.1.** Let us consider a PBO instance  $I_1$ , which consists of three hard constraints and an objective function:  $c_1 : 4\neg x_1 + x_2 + x_3 \geq 4$ ,  $c_2 : 3\neg x_1 + x_2 + 2x_3 \geq 4$ ,  $c_3 : 29x_1 + 15x_2 + 16x_3 \geq 29$ , minimize:  $x_1 + \neg x_2$ . The values of  $viol$  of hard constraints and the value of  $obj$  of the objective function under all assignments are presented in Table 1. Solutions for  $I_1$  are those resulting in the zero value of  $viol$  for all hard constraints. From Table 1, the optimal solution is  $\alpha^* = (0, 1, 1)$ , and  $obj(\alpha^*)$  is 0.

Given instance  $I_1$ , consider a scoring function without any weighting scheme, or equivalently, the weight of each constraint is 1, i.e.,  $w(c_1) = 1$ ,  $w(c_2) = 1$ ,  $w(c_3) = 1$ ,  $w(o) = 1$ ; the initial assignment  $\alpha = (0, 0, 0)$ . In accordance with the definition of the scoring function in *LS-PBO*,  $score(x_1) = 21$ ,  $score(x_2) = 17$ , and  $score(x_3) = 17$ . Actually, in order to optimize the assignment, SLS algorithms tend to select the variable to be flipped as the one with the largest score, so in this situation,  $x_1$  is picked. After flipping  $x_1$ , the assignment becomes  $\alpha = (1, 0, 0)$ , and the  $score$  value of each variable becomes  $score(x_1) = -21$ ,  $score(x_2) = 3$ ,  $score(x_3) = 3$ . Then, no matter whether  $x_2$  or  $x_3$  is flipped, the Hamming distance between

the current assignment and the optimal solution is the same as that between the initial assignment and the optimal solution, which is two. The search is not progressing in the direction towards the optimal solution.

In practice, PBO instances encoded from real-world problems are much more complex than the given illustrative example. If SLS algorithms conduct the search in incorrect directions, it would be difficult to identify a promising search space that is more likely to contain the optimal solution or those close to optimality.

As presented in Table 1, when we focus on the value of  $viol(c_3)$ , for those cases where the value of  $viol(c_3)$  is not 0, its value is much larger than the  $viol$  value of other hard constraints. Considering that each hard constraint has a *penalty* value directly proportional to its  $viol$  value, utilizing scoring function aims to guide the search towards the area with a lower sum of *penalty* values. Consequently, through making use of such scoring function, the algorithms would prefer the falsified literal with the largest coefficient to be *True* (in instance  $I_1$ , under the assumption that the current assignment is  $(0, 0, 0)$ , this falsified literal is  $x_1$  in the hard constraint  $c_3$ ).

**Our New Scoring Function.** In our opinion, a good scoring function for PBO should balance the  $viol$  values of different constraints. To this end, we propose to smooth the *penalty* values of constraints. For simplicity, we denote the smoothing function of a hard constraint  $c$  as  $smooth(c)$ , and the smoothing function of the objective function  $o$  as  $smooth(o)$ . Based on the idea of balancing the  $viol$  value, we propose the following, new scoring function:

- **For a hard constraint  $c$ , the penalty function is defined as  $penalty(c) = \frac{w(c) \times viol(c)}{smooth(c)}$** ; then the hard score of a variable  $x$  is defined as the decrement of the sum of the penalty function values of all hard constraints caused by flipping  $x$ , which is denoted by  $hscore(x)$ .
- **For the objective function  $o$ , the value of the objective function is  $obj(\alpha)$ , and the penalty function is defined as  $penalty(o) = \frac{w(o) \times obj(\alpha)}{smooth(o)}$** ; then the soft score of a variable  $x$  is defined as the decrement of the penalty function value of the objective function caused by flipping  $x$ , which is denoted by  $sscore(x)$ .
- The score of a variable  $x$  is defined as  $score(x) = hscore(x) + sscore(x)$ .

In order to instantiate the above scoring function, we propose to use a method for smoothing by using the average of the constraint coefficients, i.e.,  $smooth(c) = round(avg_{coe}(c))$ ,  $smooth(o) = round(avg_{coe}(o))$  ( $round$  is a rounding function). Consider the PBO instance  $I_1$  in Example 3.1, which has  $smooth(c_1) = 2$ ,  $smooth(c_2) = 2$ ,  $smooth(c_3) = 20$ , and  $smooth(o) = 1$ . Assume that each hard constraint and the objective function have a weight of 1 and the current assignment  $\alpha = (0, 0, 0)$ . Based on the new scoring function,  $score(x_1) = -3.05$ ,  $score(x_2) = 2.25$ , and  $score(x_3) = 1.3$ . Hence, SLS algorithms would select variable  $x_2$  to be flipped. Flipping  $x_2$  would change the current assignment  $\alpha$  to  $(0, 1, 0)$ , and the  $score$  value of each variable would become  $score(x_1) = -3.3$ ,  $score(x_2) = -2.25$ ,  $score(x_3) = 0.7$ . Afterward, SLS algorithms would select variable  $x_3$  to be flipped. If  $x_3$  is flipped, assignment  $\alpha$  becomes  $(0, 1, 1)$ , which is the optimal solution of instance  $I_1$ .

According to this illustrative example, it can be observed that, by using the average of constraint coefficients to smooth the *penalty* value, the issue of the large difference among coefficients of a variable in various constraints can be alleviated, resulting in a more effective scoring function.



### 3.2 A New Weighting Scheme

Combinatorial optimization problems with both hard and soft constraints require effective weighting schemes that balance the weights of hard and soft constraints. A potential problem was pointed out in a previous study [10]: the excessive weight given to soft constraints may make it difficult to satisfy all the hard constraints, thereby hindering the algorithm's capability of finding solutions. Moreover, an existing study [38] demonstrates that designing a weighting scheme for problems with hard constraints is challenging as it requires weighting unsatisfied constraints while maintaining the distinction between hard and soft constraints.

To alleviate the above problem, the weighting scheme proposed in *LS-PBO* sets an upper bound  $\zeta$  (an integer parameter) to the maximum value of the objective function weight. We use *unsat\_hard\_set* to denote the set of unsatisfied hard constraints. For a PBO instance  $I$ , the average of the product of the  $avg_{coe}(c)$  and  $w(c)$  of all constraints  $c$  is denoted as  $wavg_{coe}(I)$ , that is,  $wavg_{coe}(I) = (\sum_{i=1}^m (avg_{coe}(c_i) \times w(c_i))) / m$ . Assuming that the current assignment is  $\alpha$ , the best solution that has been found is  $\alpha^*$ , and its corresponding objective function value is  $obj(\alpha^*)$ .

The weighting scheme adopted in *LS-PBO* is described as follows:

- Initialization phase: at the start of the local search process, the weight of each hard constraint  $c$  is initialized as 1, i.e.,  $w(c) := 1$ ; the weight of the objective function  $o$  is also initialized as 1, i.e.,  $w(o) := 1$ .
- Update phase: when the search is trapped in a local optimum (i.e., there is no variable whose *score* value is greater than 0), for each  $c$  in *unsat\_hard\_set*,  $w(c) := w(c) + 1$ ; if  $obj(\alpha) \geq obj(\alpha^*)$  and  $w(o) \times avg_{coe}(o) - wavg_{coe}(I) \leq \zeta$ ,  $w(o) := w(o) + 1$ , where  $\zeta$  is a parameter introduced by *LS-PBO*.

In fact, the setting of  $\zeta$  greatly affects the performance of *LS-PBO*, and if the average of the coefficients of the objective function is much greater than the average of the coefficients of the hard constraints, the weight of the objective function basically would not be updated. In addition, varying the timing of weighting constraints could be a promising strategy to improve the performance of the weighting scheme.

We propose to deal with these problems by modifying the condition of updating weights. Specifically, we propose a stricter condition for increasing objective function weight. Our proposed weighting scheme is as follows:

- Initialization phase: at the start of the local search process, the weight of each hard constraint  $c$  is initialized as 1, i.e.,  $w(c) := 1$ ; **the weight of the objective function  $o$  is initialized as 0, i.e.,  $w(o) := 0$ .**
- Update phase: when the search is trapped in a local optimum (i.e., there is no variable whose *score* value is greater than 0), for each  $c$  in *unsat\_hard\_set*,  $w(c) := w(c) + 1$ ; **if *unsat\_hard\_set* is empty,  $w(o) := w(o) + 1$ .**

In the beginning, the weight of the objective function is initialized as 0, so that the algorithm would first focus on finding solutions. If the search is trapped in a local optimum, the weight of the objective function is increased only when the current assignment  $\alpha$  is a solution (all hard constraints are satisfied under  $\alpha$ ). Accordingly, if the algorithm frequently visits solutions, then the objective function would have a greater chance to increase its weight. Otherwise, there would be limited opportunities to increase the objective function weight.

■ **Algorithm 1** The *NuPBO* Algorithm.

---

**Input:** A PBO instance  $I$ , *cutoff time*.  
**Output:** The best solution ( $\alpha^*$ ) found and its objective function value  $obj^*$ , or “No solution found”.

```

1  $\alpha^* := \emptyset$ ;  $obj^* := +\infty$ ;
2 while no terminating criteria are met do
3    $\alpha :=$  an initial assignment;
4   for each  $c$  in hard constraints do
5      $w(c) := 1$ ;
6   for the objective function  $o$ ,  $w(o) := 0$ ;
7    $L := 10000000$ ;
8   for  $step = 0$ ;  $step < L$ ;  $step++$  do
9     if  $\alpha$  is feasible and  $obj^* > obj(\alpha)$  then
10       $\alpha^* := \alpha$ ;  $obj^* := obj(\alpha)$ ;  $L := step + 10000000$ ;
11      if  $D := \{x \mid score(x) > 0\} \neq \emptyset$  then
12         $v :=$  a variable in  $D$  with the highest score;
13      else
14        update constraints weights by the new weighting scheme described in Section 3.2;
15        if  $\exists$  unsatisfied hard constraints then
16           $c :=$  a random unsatisfied hard constraint;
17           $v :=$  the variable whose literal is false with highest score in  $c$ ;
18        else
19           $v :=$  a randomly chosen variable with score  $> 0$ ;
20       $\alpha := \alpha$  with  $v$  flipped;
21 if  $\alpha^* \neq \emptyset$  then return  $\alpha^*$  and  $obj^*$ ;
22 else return No solution found;

```

---

#### 4 The *NuPBO* Algorithm

In this section, we develop a new SLS algorithm named *NuPBO*, which is based on the main ideas proposed in Section 3. *NuPBO* adopts the Dynamic Local Search (DLS) framework as does *LS-PBO*. The pseudo-code of *NuPBO* is outlined in Algorithm 1. We use  $\alpha^*$  and  $obj^*$  to denote the best-found solution and the corresponding objective function value (i.e., the cost of the best-found solution), while  $\alpha$  denotes the current assignment which is maintained during the search.

In the beginning,  $\alpha^*$  is initialized as an empty set, and  $obj^*$  is initialized as  $+\infty$  (Line 1). *NuPBO* then iteratively calls the local search process until reaching a terminating criterion (e.g., reaching a preset cutoff time, or achieving a feasible assignment  $\alpha$  whose corresponding  $obj$  value is equal to 0) (Lines 2–20).

In the local search process, an initial assignment is generated by assigning each Boolean variable to a default value 0 (as in *LS-PBO*) (Line 3). *NuPBO* then initializes the weights of all hard constraints as 1, and sets the weight of objective function to 0 according to our proposed weighting scheme. After initialization, *NuPBO* conducts the search process (Lines 8–20). During the search process, whenever *NuPBO* finds a solution whose  $obj$  is lower than  $obj^*$ , then  $\alpha^*$  and  $obj^*$  are updated accordingly.

In each search step, *NuPBO* selects a variable and flips it based on two situations: (I) If the set  $D$  of decreasing variables (i.e., the variable  $x$  with  $score(x) > 0$ ) is not empty, a variable with the highest score is selected from  $D$ , breaking ties by preferring the variable



that has been flipped least recently. (II) When  $D$  is empty, which indicates that the search is trapped in a local optimum, then *NuPBO* updates the weights of constraints according to our new weighting scheme. Then, if there exist unsatisfied hard constraints, an unsatisfied hard constraint  $c$  is randomly picked. As we know, for a MaxSAT instance, if a clause  $cm$  is unsatisfied, then it would become satisfied after flipping any variables in  $cm$ . For a PBO instance, if a hard constraint  $cp$  is unsatisfied (i.e.,  $viol(cp) > 0$ ), only flipping variables whose literals are *False* under the current assignment can reduce  $viol(cp)$  (Assuming under the current assignment  $\alpha$ ,  $x_1$  is 1, then the literal  $x_1$  is *True*, while the literal  $\neg x_1$  is *False*). Therefore, *NuPBO* picks the variable whose literal is *False* with the highest *score* in  $c$ . Otherwise (i.e., all the hard constraints are satisfied), *NuPBO* randomly chooses a variable whose *sscore* is greater than 0.

Finally, when any terminating criterion is met, *NuPBO* stops and reports the best solution  $\alpha^*$  and  $obj^*$  if a solution is found; otherwise, it reports “No solution found”.

## 5 Experimental Evaluations

In this section, we introduce experimental preliminaries and then conduct extensive experiments on 6 PBO benchmarks. First, we compare *NuPBO* with 5 state-of-the-art PBO solvers. Second, we conduct experiments to show that combining *NuPBO* with complete solvers can lead to better portfolios. Third, we report experimental results to demonstrate the effectiveness of our main ideas. Finally, we examine the stability of the SLS solvers by running each SLS solver 10 times with seeds ranging from 1 to 10.

### 5.1 Experimental Preliminaries

**Benchmarks.** We evaluate *NuPBO* on 6 benchmarks, which are described as follows:

- **PB16:** the OPT-SMALLINT-LIN benchmark from the latest 2016 pseudo-Boolean competition. As a mainstream benchmark for evaluating the performance of PBO solvers, it consists of 1600 instances of various categories.<sup>2</sup>
- **MIPLIB:** 0-1 integer linear programming optimization problems. This benchmark contains 291 instances of various categories, provided in the literature [12].<sup>3</sup>
- **CRAFT:** crafted combinatorial benchmarks whose coefficients are small integers. This benchmark contains 955 instances of various categories, provided in the literature [12].<sup>4</sup>
- **MWCB:** the Minimum-Width Confidence Band Problem. This benchmark contains 24 instances.
- **SAP:** the Seating Arrangements Problem. This benchmark contains 21 instances.
- **WSNO:** the Wireless Sensor Network Optimization Problem. This benchmark consists of 18 instances.

For the benchmarks of **MWCB**, **SAP**, and **WSNO**, the descriptions, the downloading websites, and the methods of converting the real-world applications into PBO instances and the encoded PBO instances are presented in the literature [26].<sup>5</sup>

<sup>2</sup> <http://www.cril.univ-artois.fr/PB16/bench/PB16-used.tar>

<sup>3</sup> <https://zenodo.org/record/3870965>

<sup>4</sup> <https://zenodo.org/record/4036016>

<sup>5</sup> <https://lcs.ios.ac.cn/%7ecaisw/Resource/LS-PBO/>

**State-of-the-Art Competitors.** We compare *NuPBO* with 5 state-of-the-art solvers, including one SLS solver (i.e., *LS-PBO*) and 4 complete solvers. The 4 complete solvers include 2 PB solvers (i.e., *PBO-IHS* and *RoundingSat*) and 2 MIP solvers (i.e., *Gurobi* and *SCIP*):

- *LS-PBO* [26]: the state-of-the-art SLS algorithm for solving PBO. Adopt the parameter setting recommended by its authors. It outperforms *Gurobi* and *RoundingSat* on many real-world application benchmarks.<sup>5</sup>
- *PBO-IHS* [37]: a recent IHS PBO solver building upon *RoundingSat* [14].<sup>6</sup>
- *RoundingSat* [12]: a recent PBO solver combining core-guided search with cutting planes reasoning.<sup>7</sup>
- *Gurobi* [19]: one of the most powerful commercial MIP solvers (Version 9.1.2). The default configuration is used, along with a single thread.<sup>8</sup>
- *SCIP* [16]: one of the fastest non-commercial solvers for MIP (Version 7.0.3, using *SoPlex* 5.0.2 as its internal LP solver).<sup>9</sup>

**Experimental Setup.** *LS-PBO* and *NuPBO* are implemented in C++, and compiled with g++ (version 8.5.0) using the option “-O3”. Installation procedures for other solvers follow their detailed guidelines. All the experiments are carried out on a workstation under the operating system CentOS, with the AMD EPYC7702 2.0GHz CPU.

In these experiments, we adopt two cutoff times of 300 CPU seconds (300s) and 3600 CPU seconds (1h). Each solver performs one run within a given cutoff time on each instance, and we record the cost of the best solution found by solver  $S_j$  on instance  $I_k$ , denoted as  $sol_{S_j I_k}$ . The cost of the best solution found among all solvers in the same table within the same cutoff time on instance  $I_k$  is denoted as  $best_{I_k}$ . For each solver  $S$  solving a benchmark  $B_i$  within a cutoff time, we use 3 *metrics* to evaluate the performance of  $S$ .

- *#win.*: the number of instances where the corresponding  $best_{I_k}$  can be obtained by solver  $S$  on  $B_i$  (i.e., the number of winning instances).
- *avg<sub>score</sub>*: in our experiments, the competition score of solver  $S_j$  on instance  $I_k$  is represented by  $score_{S_j I_k} = \frac{best_{I_k} + 1}{sol_{S_j I_k} + 1}$ , which measures the gap between  $sol_{S_j I_k}$  and  $best_{I_k}$ . If solver  $S_j$  could not report a solution on instance  $I_k$ , then  $score_{S_j I_k} = 0$ . We use *avg<sub>score</sub>* to denote the average competition score of a solver on a benchmark. The competition score of each solver on each instance is the metric to measure the performance of solvers in the incomplete track of recent MaxSAT Evaluations (2017-2023).
- *#feas.*: the number of instances where solver  $S$  obtains solutions on  $B_i$ .

In our experiments, *avg<sub>score</sub>* is calculated by ignoring the instances that are proven to have no solution by the complete solvers. Based on the preliminary experiments, we conclude that at least 123 instances in the PB16 benchmark and at least 17 instances in the MIPLIB benchmark do not have solutions. All instances in the CRAFT, MWCB, SAP, and WSNO benchmarks have solutions.<sup>10</sup>

The number of instances in each benchmark is indicated by ‘#inst.’. For each of the above three metrics, if a solver obtains a larger metric value on a benchmark, then the solver exhibits better performance on the benchmark. The results highlighted in **bold** indicate the best performance for the corresponding metric.

<sup>6</sup> <https://bitbucket.org/coreo-group/pbo-ihs-solver/>

<sup>7</sup> <https://doi.org/10.5281/zenodo.4043124>

<sup>8</sup> <https://www.gurobi.com/products/gurobi-optimizer/>

<sup>9</sup> <https://www.scipopt.org/index.php#download>

<sup>10</sup> Note that the definition of the competition *score* (metric) in this subsection has no relationship to the definition of the *score* in the scoring function in subsection 3.1.

■ **Table 2** Experimental results of *NuPBO* and all the competitors on all the benchmarks (top: cutoff 300s, bottom: cutoff 1h) (Benc, i.e., Benchmark. *avg<sub>s</sub>*, i.e., *avg<sub>score</sub>*).

Benc	#inst.	SLS solvers				PB solvers				MIP solvers			
		<i>NuPBO</i>		<i>LS-PBO</i>		<i>PBO-IHS</i>		<i>RoundingSat</i>		<i>Gurobi</i>		<i>SCIP</i>	
		#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>	#win. #feas.	<i>avg<sub>s</sub></i>
<i>cutoff 300s</i>													
PB16	1600	1049 1351	0.8800	700 1183	0.7516	930 <b>1401</b>	0.8712	964 1392	0.8916	<b>1158</b> 1365	<b>0.9011</b>	887 1244	0.7918
MIPLIB	291	130 242	<b>0.8480</b>	75 222	0.7375	84 230	0.7350	89 <b>245</b>	0.7855	<b>166</b> 235	0.8013	114 221	0.7360
CRAFT	955	<b>894</b> 943	0.9868	816 930	0.9682	809 930	0.9433	825 936	0.9639	893 <b>955</b>	<b>0.9961</b>	764 921	0.9583
MWCB	24	<b>24</b> <b>24</b>	<b>1.0000</b>	0 <b>24</b>	0.9448	0 19	0.4496	0 <b>24</b>	0.6247	0 12	0.4004	0 3	0.0761
SAP	21	<b>21</b> <b>21</b>	<b>1.0000</b>	0 <b>21</b>	0.9750	0 0	0.0000	0 0	0.0000	0 1	0.0395	0 0	0.0000
WSNO	18	<b>18</b> <b>18</b>	<b>1.0000</b>	11 <b>18</b>	0.9026	2 5	0.1738	4 <b>18</b>	0.6624	4 5	0.2431	0 6	0.1631
<i>cutoff 1h</i>													
PB16	1600	1064 1360	0.8897	814 1278	0.8164	990 <b>1412</b>	0.8875	1012 1401	0.9100	<b>1229</b> 1396	<b>0.9354</b>	1012 1304	0.8565
MIPLIB	291	127 243	0.8519	83 231	0.7673	100 239	0.7903	87 247	0.8099	<b>199</b> <b>253</b>	<b>0.9023</b>	142 238	0.8104
CRAFT	955	891 <b>955</b>	<b>0.9992</b>	844 932	0.9714	886 947	0.9841	873 <b>955</b>	0.9902	<b>930</b> <b>955</b>	0.9987	824 930	0.9704
MWCB	24	<b>23</b> <b>24</b>	<b>0.9998</b>	1 <b>24</b>	0.9690	0 <b>24</b>	0.5620	0 <b>24</b>	0.7116	0 <b>24</b>	0.7437	0 17	0.5058
SAP	21	<b>21</b> <b>21</b>	<b>1.0000</b>	0 <b>21</b>	0.9785	0 0	0.0000	0 0	0.0000	0 1	0.0451	0 0	0.0000
WSNO	18	<b>18</b> <b>18</b>	<b>1.0000</b>	15 <b>18</b>	0.9985	5 14	0.5989	11 <b>18</b>	0.8660	4 13	0.4904	0 4	0.0842

## 5.2 Comparisons with State-of-the-Art Solvers

The comparative results of *NuPBO* and all the competitors on all the benchmarks are shown in Table 2. We first analyze the results with a cutoff time of 300s.

- In terms of the number of winning instances, *NuPBO* gives the best performance on 4 benchmarks, including CRAFT and the 3 real-world application benchmarks, and ranked second on the PB16 and MIPLIB benchmarks (with *Gurobi* being the best).
- In terms of *avg<sub>score</sub>*, *NuPBO* outperforms all the competitors on 4 benchmarks, including MIPLIB and the 3 real-world application benchmarks. On the PB16 benchmark, its *avg<sub>score</sub>* ranks third after *Gurobi* and *RoundingSat*. The *avg<sub>score</sub>* value of *NuPBO* ranks second on the CRAFT benchmark behind *Gurobi*.
- In terms *#feas.*, *NuPBO* and *LS-PBO* find solutions for all instances in the 3 real-world application benchmarks, while *PBO-IHS*, *RoundingSat*, and *Gurobi* are respectively the best for finding solutions on PB16, MIPLIB, and CRAFT. Although the value of *#feas.* of *NuPBO* ranks second on MIPLIB and CRAFT benchmarks, and ranks fourth on PB16 benchmark, *NuPBO* performs considerably better than *LS-PBO*, an SLS solver for PBO.

■ **Table 3** Experimental results of  $VBS_{all}$ ,  $VBS_{exclude\_lspbo}$ , and  $VBS_{exclude\_nupbo}$  on all the benchmarks (top: cutoff 300s, bottom: cutoff 1h).

Benchmark	#inst.	$VBS_{all}$			$VBS_{exclude\_lspbo}$			$VBS_{exclude\_nupbo}$		
		#win.	avg <sub>score</sub>	#feas.	#win.	avg <sub>score</sub>	#feas.	#win.	avg <sub>score</sub>	#feas.
<i>cutoff 300s</i>										
PB16	1600	<b>1434</b>	<b>0.9709</b>	<b>1434</b>	1430	0.9703	<b>1434</b>	1356	0.9690	1433
MIPLIB	291	<b>254</b>	<b>0.9270</b>	254	<b>254</b>	<b>0.9270</b>	254	218	0.9119	254
CRAFT	955	<b>955</b>	<b>1.0000</b>	955	<b>955</b>	<b>1.0000</b>	955	939	0.9999	955
MWCB	24	<b>24</b>	<b>1.0000</b>	24	<b>24</b>	<b>1.0000</b>	24	0	0.9448	24
SAP	21	<b>21</b>	<b>1.0000</b>	21	<b>21</b>	<b>1.0000</b>	21	0	0.9750	21
WSNO	18	<b>18</b>	<b>1.0000</b>	18	<b>18</b>	<b>1.0000</b>	18	11	0.9032	18
<i>cutoff 1h</i>										
PB16	1600	<b>1440</b>	<b>0.9749</b>	<b>1440</b>	1438	0.9747	<b>1440</b>	1394	0.9730	1438
MIPLIB	291	<b>262</b>	<b>0.9562</b>	262	<b>262</b>	<b>0.9562</b>	262	238	0.9492	262
CRAFT	955	<b>955</b>	<b>1.0000</b>	955	<b>955</b>	<b>1.0000</b>	955	953	>0.9999	955
MWCB	24	<b>24</b>	<b>1.0000</b>	24	23	0.9998	24	1	0.9690	24
SAP	21	<b>21</b>	<b>1.0000</b>	21	<b>21</b>	<b>1.0000</b>	21	0	0.9785	21
WSNO	18	<b>18</b>	<b>1.0000</b>	18	<b>18</b>	<b>1.0000</b>	18	15	0.9985	18

With the cutoff time of 1h,  $NuPBO$  outperforms  $LS-PBO$  on the 3 real-world application benchmarks. On the other 3 benchmarks,  $NuPBO$  shows competitive performance compared to  $Gurobi$ , and significantly outperforms the state-of-the-art SLS solver  $LS-PBO$  in terms of all metrics of #win., avg<sub>score</sub>, and #feas..

### 5.3 Complementarity between SLS Solvers and Complete Solvers

In this subsection, we conduct experiments to investigate the complementarity between SLS solvers and complete solvers when solving PBO.

To investigate the complementarity between SLS solvers and complete solvers, we construct three perfect portfolio selectors: given a set of base solvers  $\Theta$ , for each instance, the solution of the perfect portfolio selector constructed on  $\Theta$  is the best among the entire collection of solutions reported by all solvers in  $\Theta$ . These three perfect portfolio selectors are built based on  $\Theta_1 = \{LS-PBO, NuPBO, PBO-IHS, RoundingSat, Gurobi, SCIP\}$ ,  $\Theta_2 = \{NuPBO, PBO-IHS, RoundingSat, Gurobi, SCIP\}$ , and  $\Theta_3 = \{LS-PBO, PBO-IHS, RoundingSat, Gurobi, SCIP\}$  dubbed  $VBS_{all}$ ,  $VBS_{exclude\_lspbo}$  and  $VBS_{exclude\_nupbo}$ , respectively. Then we conduct experiments to evaluate the performance of these three perfect portfolio selectors on all benchmarks. The related results are presented in Table 3.

The comparison between  $VBS_{all}$  and  $VBS_{exclude\_lspbo}$  reveals the number of instances where only the LS-PBO solver can achieve the optimal solution among all solvers. Similarly, comparing  $VBS_{all}$  and  $VBS_{exclude\_nupbo}$  shows the number of instances where only the NuPBO solver can obtain the optimal solution among all solvers. As shown in Table 3, taking the PB16 benchmark with a cutoff time of 300 seconds as an example, out of the 1600 instances, there are 4 instances where only LS-PBO can achieve the optimal solution among all solvers, and there are 78 instances where only NuPBO can obtain the optimal solution among all solvers. Additionally, in 1 instance, only NuPBO was able to find a feasible solution. The results in Table 3 demonstrate that, compared to the state-of-the-art SLS solver  $LS-PBO$ ,  $NuPBO$  is able to enhance the complementarity between SLS solvers and complete solvers, which indicates that a portfolio selector, which combines  $NuPBO$  and complete solvers, could advance the state of the art in PBO solving.

■ **Table 4** Experimental results of *NuPBO*, *NuPBO-alt(s)*, and *NuPBO-alt(w)* on all the benchmarks (top: cutoff 300s, bottom: cutoff 1h).

Benchmark	#inst.	<i>NuPBO</i>			<i>NuPBO-alt(s)</i>			<i>NuPBO-alt(w)</i>		
		#win.	avg <sub>score</sub>	#feas.	#win.	avg <sub>score</sub>	#feas.	#win.	avg <sub>score</sub>	#feas.
<i>cutoff 300s</i>										
PB16	1600	<b>1141</b>	<b>0.9026</b>	<b>1351</b>	1024	0.8288	1253	1034	0.8822	1323
MIPLIB	291	<b>182</b>	<b>0.8687</b>	<b>242</b>	147	0.8347	<b>242</b>	112	0.8312	239
CRAFT	955	<b>941</b>	<b>0.9874</b>	<b>943</b>	<b>941</b>	0.9864	942	848	0.9648	925
MWCB	24	<b>24</b>	<b>1.0000</b>	24	0	0.9466	24	0	0.9700	24
SAP	21	<b>13</b>	<b>0.9990</b>	21	9	0.9974	21	0	0.9782	21
WSNO	18	17	0.9995	18	<b>18</b>	<b>1.0000</b>	18	15	0.9704	18
<i>cutoff 1h</i>										
PB16	1600	<b>1170</b>	<b>0.9076</b>	<b>1360</b>	1113	0.8842	1329	1090	0.8911	1336
MIPLIB	291	<b>188</b>	<b>0.8754</b>	243	149	0.8520	244	121	0.8620	<b>245</b>
CRAFT	955	<b>953</b>	<b>&gt;0.9999</b>	<b>955</b>	<b>953</b>	<b>&gt;0.9999</b>	<b>955</b>	864	0.9683	926
MWCB	24	<b>24</b>	<b>1.0000</b>	24	0	0.9459	24	0	0.9655	24
SAP	21	11	0.9986	21	<b>13</b>	<b>0.9988</b>	21	0	0.9824	21
WSNO	18	18	1.0000	18	18	1.0000	18	18	1.0000	18

## 5.4 Analysis on the Underlying Ideas

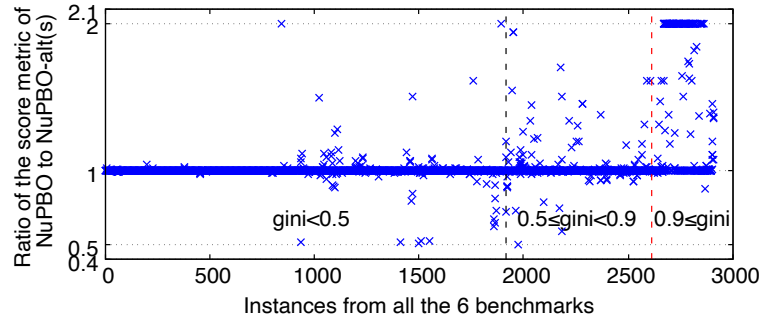
In order to demonstrate the effectiveness of our two main ideas in our *NuPBO* solver, we conduct comparative experiments on 3 solvers. We develop two alternative versions of *NuPBO*, by replacing its scoring function and weighting scheme with the ones proposed in *LS-PBO*, dubbed *NuPBO-alt(s)* and *NuPBO-alt(w)*, respectively. The weighting scheme proposed in *LS-PBO* introduces a parameter  $\zeta$  for *NuPBO-alt(w)*, which is set to 100 as recommended by *LS-PBO*'s authors [26].

The comparative results of the cutoff time of 300 seconds and 1 hour are shown in Table 4. From Table 4, *NuPBO* outperforms its alternative versions on the majority of instances. We first discuss the effectiveness of the new scoring function.

**Regarding the New Scoring Function.** With the cutoff time of 300s, in terms of the metrics of #win. and avg<sub>score</sub>, *NuPBO* exhibits the best performance on 5 benchmarks. In terms of the metric of #feas., *NuPBO* performs better than *NuPBO-alt(s)* on the PB16 benchmark and CRAFT benchmark. On the remaining 4 benchmarks, the number of feasible solutions obtained by *NuPBO* is equal to that obtained by *NuPBO-alt(s)*.

With the cutoff time of 1h, *NuPBO* exhibits significantly better performance than *NuPBO-alt(s)* on 3 benchmarks including PB16, MIPLIB, and MWCB. On the remaining 3 benchmarks, namely CRAFT, SAP and WSNO, the performance of *NuPBO* is comparable to that of *NuPBO-alt(s)*.

To examine the intuition in Section 3.1, we conduct an experiment to analyze the relationship between the instance feature and the performance difference between *NuPBO* and *NuPBO-alt(s)*. Due to the difficulty of counting the coefficients of all variables in different constraints within an instance, we use the Gini coefficient [18] of the degree of hard constraints as the instance feature, denoted by  $Gini_d$ . For a PBO instance  $I_1$ , if the degree values of all hard constraints in  $I_1$  are arranged in ascending order,  $Gini_d$  can be calculated as follows:  $Gini_d = \frac{2}{n^2 \bar{d}} \sum_{i=1}^n i(d_i - \bar{d})$ , where  $n$  is the number of hard constraints,  $i$  is the



■ **Figure 1** The ratio of the *score* metric of *NuPBO* and *NuPBO-alt(s)* on instances from all the 6 benchmarks (cutoff 300s).

rank of degree values in ascending order,  $d_i$  is the degree of  $i$ -th hard constraint ( $d_i$  values are in ascending order), and  $\bar{d}$  is the mean value.<sup>11</sup> The greater  $Gini_d(I_1)$ , the greater the inequality between the degrees of constraints in instance  $I_1$ . In those instances whose  $Gini_d$  is large, the coefficient of a variable may differ greatly between constraints. For an instance  $I_1$ , we use  $score_{NuPBO I_1}$  to represent the competition score of *NuPBO*, and  $score_{NuPBO-alt(s) I_1}$  to represent the competition score of *NuPBO-alt(s)*.  $R(I_1) = \frac{score_{NuPBO I_1} + 1}{score_{NuPBO-alt(s) I_1} + 1}$  is used to denote the performance difference between *NuPBO* and *NuPBO-alt(s)*. Thus, if *NuPBO* finds a solution while *NuPBO-alt(s)* does not,  $R = 2$  (on the contrary,  $R = 0.5$ ). If  $R = 1$ , *NuPBO* and *NuPBO-alt(s)* obtained the same competition score (or no solution has been found).

We conduct an experiment on all 6 benchmarks with a cutoff time of 300s. The related results are presented in Figure 1. According to Figure 1, the x-axis represents 2909 instances of the 6 benchmarks, sorted by  $Gini_d$  in ascending order, and the y-axis represents the corresponding  $R$  values.

Results in Figure 1 demonstrate that *NuPBO* outperforms *NuPBO-alt(s)*, as the number of instances with  $R > 1$  exceeds those with  $R < 1$ . In addition, on instances with  $Gini_d \geq 0.9$ , *NuPBO* exhibits a significant performance advantage over *NuPBO-alt(s)*, and many instances in this category have an  $R$  value of 2, which indicates that *NuPBO* performs much better in terms of the metric of  $\#feas.$ . On instances with  $Gini_d \geq 0.5$ , *NuPBO* also shows performance improvement over *NuPBO-alt(s)*.

**Regarding the New Weighting Scheme.** With the cutoff time of 300s, in terms of the metrics of  $\#win.$  and  $avg_{score}$ , *NuPBO* outperforms *NuPBO-alt(w)* on all the benchmarks. In terms of the metric of  $\#feas.$ , *NuPBO* achieves better performance than *NuPBO-alt(w)* on 3 benchmarks. On the other 3 benchmarks, the value of  $\#feas.$  achieved by *NuPBO* is equal to that obtained by *NuPBO-alt(w)*.

With the cutoff time of 1h, regarding the metrics of  $\#win.$  and  $avg_{score}$ , *NuPBO* outperforms *NuPBO-alt(w)* on 5 out of 6 benchmarks, and achieves the same performance on the WSNO benchmark. Regarding the metric of  $\#feas.$ , *NuPBO* demonstrates better performance than *NuPBO-alt(w)* on 2 benchmarks. On the MIPLIB benchmark, the performance of *NuPBO-alt(w)* is only slightly better than that of *NuPBO*. On the 2 real-world application benchmarks, these SLS solvers achieve the same performance. The experimental results clearly indicate the effectiveness of our proposed new weighting scheme.

<sup>11</sup> [https://www.statsdirect.com/help/default.htm#nonparametric\\_methods/gini.htm](https://www.statsdirect.com/help/default.htm#nonparametric_methods/gini.htm)

■ **Table 5** Experimental results of *NuPBO*, *LS-PBO*, *NuPBO-alt(s)*, and *NuPBO-alt(w)* with seeds ranging from 1 to 10 on all the benchmarks (left: cutoff 300s, right: cutoff 1h).

Benchmark	#inst.	cutoff 300s			cutoff 1h		
		$avg_{avgsol}$	$avg_{stdev}$	$\frac{avg_{stdev}}{avg_{avgsol}}$	$avg_{avgsol}$	$avg_{stdev}$	$\frac{avg_{stdev}}{avg_{avgsol}}$
<i>NuPBO</i>							
PB16	1600	34567.32	259.79	0.75%	34721.64	123.30	0.36%
CRAFT	955	3035410.02	0.10	<0.01%	3000465.70	0.09	<0.01%
MIPLIB	291	59271045.04	240944.55	0.41%	58116554.74	299542.17	0.52%
MWCB	24	197890.62	1504.80	0.76%	193513.34	888.91	0.46%
SAP	21	1039.04	3.74	0.36%	1033.64	3.00	0.29%
WSNO	18	1301.21	225.55	17.33%	1158.61	0.00	0.00%
<i>LS-PBO</i>							
PB16	1600	30844.62	143.86	0.47%	34305.81	213.45	0.62%
CRAFT	955	3074484.99	1.69	<0.01%	3071190.03	1.11	<0.01%
MIPLIB	291	53327323.12	1262751.16	2.37%	50874231.31	924287.05	1.82%
MWCB	24	209821.48	1582.87	0.75%	201482.90	1525.52	0.76%
SAP	21	1066.74	4.61	0.43%	1059.17	3.43	0.32%
WSNO	18	1448.88	299.06	20.64%	1174.76	44.64	3.80%
<i>NuPBO-alt(s)</i>							
PB16	1600	36842.12	112.50	0.31%	35779.84	129.39	0.36%
CRAFT	955	3038627.52	0.10	<0.01%	3000465.70	0.10	<0.01%
MIPLIB	291	65612369.74	911077.41	1.39%	63282671.88	90491.84	0.14%
MWCB	24	210377.43	1688.62	0.80%	205368.10	1267.23	0.62%
SAP	21	1039.50	3.49	0.34%	1034.15	2.68	0.26%
WSNO	18	1295.50	192.98	14.90%	1158.65	0.12	0.01%
<i>NuPBO-alt(w)</i>							
PB16	1600	34535.79	312.20	0.90%	34594.61	214.89	0.62%
CRAFT	955	3094400.67	1.45	<0.01%	3094398.61	1.15	<0.01%
MIPLIB	291	59063484.68	259177.44	0.44%	57971941.39	350572.11	0.60%
MWCB	24	205188.29	1467.44	0.72%	201127.28	929.57	0.46%
SAP	21	1061.47	4.08	0.38%	1054.01	3.41	0.32%
WSNO	18	1293.97	143.69	11.10%	1159.25	2.02	0.17%

## 5.5 Stability of Local Search Solvers

In order to examine the stability of all four SLS solvers adopted in our experiments, each of the four SLS solvers runs 10 times with seeds ranging from 1 to 10 on all instances from all 6 benchmarks.

For a given solver  $S$  and an instance  $I$ :  $sol_{SIJ}$  denotes the cost of the best solution found by solver  $S$  with seed  $J$  on instance  $I$ ,  $avgsol$  denotes the average cost of best solutions obtained by solver  $S$  over all 10 runs on instance  $I$ , while  $stdev$  denotes the standard deviation of the cost of the best solutions obtained by solver  $S$  over all 10 runs on instance  $I$ . On a benchmark  $B$  consisting of multiple instances:  $avg_{avgsol}$  represents the average value of  $avgsol$  obtained by solver  $S$  over all instances where solutions are obtained, while  $avg_{stdev}$  stands for the average value of  $stdev$  obtained by solver  $S$  over all instances where solutions are found. The calculation of  $avg_{avgsol}$  is based on instances where solutions are found,



different solvers may find solutions on different subsets of instances for a given benchmark and cutoff time. In addition, for a given benchmark, it is possible that a solver finds solutions on more instances within a cutoff time of 1h than adopting a cutoff time of 300s. Moreover, according to the above definition of  $avg_{avg_{sol}}$ , we note that the value of  $avg_{avg_{sol}}$  cannot be used to compare the performance of different solvers.

The experimental results presented in Table 5 demonstrate that, with the cutoff time of 300s, all four SLS solvers exhibit stable performance on 5 out of 6 benchmarks, while on the WSNO benchmark, the performance is less stable compared to the other benchmarks. In addition, the values of  $\frac{avg_{stdev}}{avg_{avg_{sol}}}$  for *NuPBO* are less than 1% on all 5 benchmarks, which clearly indicates that *NuPBO* can achieve stable performance. With the cutoff time of 1h, all four SLS solvers perform stably on the 6 benchmarks.

## 6 Conclusions and Future Work

This paper is devoted to improving the performance of SLS solvers for solving the PBO problem via a new scoring function and a new weighting scheme. First, we introduced our new scoring function. Furthermore, we proposed a new weighting scheme that effectively determines when to increase the weight of the objective function. Based on these two main ideas, we developed a new SLS solver named *NuPBO*. Extensive experimental results demonstrate that *NuPBO* significantly outperforms *LS-PBO* on all testing benchmarks. *NuPBO* outperforms all its competitors on 3 real-world application benchmarks and shows competitive performance compared to state-of-the-art competitors on solving PB16, MIPLIB, and CRAFT benchmarks. In addition, *NuPBO* enhances the complementarity between SLS solvers and complete solvers on all testing benchmarks.

For future work, we would like to develop more efficient heuristic strategies and explore the effect of instance features on the performances of different categories of PBO solvers.

---

## References

- 1 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.
- 2 Carlos Ansótegui and Joel Gabàs. WPM3: An (in)complete algorithm for weighted partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017.
- 3 VL Beresnev, EN Goncharov, and AA Mel’nikov. Local search with a generalized neighborhood in the optimization problem for pseudo-boolean functions. *Journal of Applied and Industrial Mathematics*, 6:22–30, 2012.
- 4 Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete MaxSAT. In *Proceedings of CPAIOR 2019*, pages 39–56, 2019.
- 5 Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.
- 6 Shaowei Cai, Chuan Luo, Jinkun Lin, and Kaile Su. New local search methods for partial MaxSAT. *Artificial Intelligence*, 240:1–18, 2016.
- 7 Shaowei Cai, Chuan Luo, and Kaile Su. Scoring functions based on second level score for k-sat with long clauses. *J. Artif. Intell. Res.*, 51:413–441, 2014. doi:10.1613/jair.4480.
- 8 Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial maxsat. In *Proceedings of AAAI 2014*, pages 2623–2629, 2014.
- 9 Byungki Cha and Kazuo Iwama. Performance test of local search algorithms using new types of random CNF formulas. In *Proceedings of IJCAI 1995*, pages 304–311, 1995.
- 10 Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial MAXSAT. In *Proceedings of AAAI 1997*, pages 263–268, 1997.

- 11 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proceedings of CP 2011*, pages 225–239, 2011.
- 12 Jo Devriendt, Stephan Gocht, Emir Demirovic, Jakob Nordström, and Peter J. Stuckey. Cutting to the core of pseudo-boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of AAAI 2021*, pages 3750–3758, 2021.
- 13 Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.
- 14 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In Jérôme Lang, editor, *Proceedings of IJCAI 2018*, pages 1291–1299, 2018.
- 15 Jan Elffers and Jakob Nordström. A cardinal improvement to pseudo-Boolean solving. In *Proceedings of AAAI 2020*, pages 1495–1503, 2020.
- 16 Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schläpfer, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. Technical report, Optimization Online, March 2020. URL: [http://www.optimization-online.org/DB\\_HTML/2020/03/7705.html](http://www.optimization-online.org/DB_HTML/2020/03/7705.html).
- 17 Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012.
- 18 Corrado Gini. Concentration and dependency ratios. *Rivista di politica economica*, pages 769–792, 1997.
- 19 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL: <https://www.gurobi.com>.
- 20 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.
- 21 Pascal Van Hentenryck and Laurent Michel. *Constraint-based local search*. The MIT press, 2009.
- 22 Holger H. Hoos, Laetitia Jourdan, Marie-Eléonore Kessaci, Thomas Stützle, and Nadarajan Veerapen. Special issue on "stochastic local search: Recent developments and trends". *International Transactions in Operational Research*, 27(1):697–698, 2020.
- 23 Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1-2):95–100, 2012.
- 24 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010.
- 25 Zhendong Lei and Shaowei Cai. Solving (weighted) partial MaxSAT by dynamic local search for SAT. In *Proceedings of IJCAI 2018*, pages 1346–1352, 2018.
- 26 Zhendong Lei, Shaowei Cai, Chuan Luo, and Holger H. Hoos. Efficient local search for pseudo boolean optimization. In *Proceedings of SAT 2021*, pages 332–348, 2021.
- 27 Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243:26–44, 2017.
- 28 Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. CCLS: An efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7):1830–1843, 2015.
- 29 Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular maxsat solver,. In *Proceedings of SAT 2014*, pages 438–445, 2014.
- 30 Rafiq Muhammad and Peter J. Stuckey. A stochastic non-CNF SAT solver. In *Proceedings of PRICAI 2006*, pages 120–129, 2006.
- 31 Alexander Nadel. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *Proceedings of FMCAD 2019*, pages 193–202, 2019.

- 32 Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proceedings of AAAI 2014*, pages 2717–2723, 2014.
- 33 Olivier Roussel and Vasco Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of satisfiability*, pages 1087–1129. IOS Press, 2021.
- 34 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 & Systems*, 98-D(6):1121–1127, 2015.
- 35 Bart Selman and Henry Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. In *Proceedings of IJCAI 1993*, pages 290–295, 1993.
- 36 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-boolean optimization by implicit hitting sets. In *Proceedings of CP 2021*, pages 51:1–51:20, 2021.
- 37 Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Improvements to the implicit hitting set approach to pseudo-boolean optimization. In *Proceedings of SAT 2022*, pages 13:1–13:18, 2022.
- 38 John Thornton and Abdul Sattar. Dynamic constraint weighting for over-constrained problems. In *Proceedings of PRICAI 1998*, pages 377–388, 1998.
- 39 Renato Tinós, Michal W Przewozniczek, and Darrell Whitley. Iterated local search with perturbation based on variables interaction for pseudo-boolean optimization. In *Proceedings of GECCO 2022*, pages 296–304, 2022.
- 40 Chris Voudouris and Edward Tsang. Partial constraint satisfaction problems and guided local search. *Proc., Practical Application of Constraint Technology (PACT'96), London*, pages 337–356, 1996.
- 41 Christos Voudouris, Edward PK Tsang, and Abdullah Alsheddy. Guided local search. In *Handbook of metaheuristics*, pages 321–361. Springer, 2010.
- 42 Joachim P. Walser. Solving linear pseudo-boolean constraint problems with local search. In *Proceedings of AAAI 1997*, pages 269–274, 1997.
- 43 Robert Wille, Hongyan Zhang, and Rolf Drechsler. ATPG for reversible circuits using simulation, boolean satisfiability, and pseudo boolean optimization. In *Proceedings of ISVLSI 2011*, pages 120–125, 2011.
- 44 Yuhang Zhang, Richard Hartley, John Mashford, and Stewart Burn. Superpixels via pseudo-boolean optimization. In *Proceedings of ICCV 2011*, pages 1387–1394, 2011.
- 45 Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, Chu-Min Li, and Felip Manyà. BandMaxSAT: A local search maxsat solver with multi-armed bandit. In *Proceedings of IJCAI 2022*, pages 1901–1907, 2022.