Deep Cooperation of Local Search and Unit Propagation Techniques

Xiamin Chen
Shanghai University of Finance and Economics, China

Zhendong Lei[1]
Huawei Taylor Lab, Shanghai, China

Pinyan Lu[2]
Shanghai University of Finance and Economics, Shanghai, China
Huawei Taylor Lab, Shanghai, China

Abstract

Local search (LS) is an efficient method for solving combinatorial optimization problems such as MaxSAT and Pseudo Boolean Problems (PBO). However, due to a lack of reasoning power and global information, LS methods get stuck at local optima easily. In contrast to the LS, Systematic Search utilizes unit propagation and clause learning techniques with strong reasoning capabilities to avoid falling into local optima. Nevertheless, the complete search is generally time-consuming to obtain a global optimal solution. This work proposes a deep cooperation framework combining local search and unit propagation to address their inherent disadvantages. First, we design a mechanism to detect when LS gets stuck, and then a well-designed unit propagation procedure is called upon to help escape the local optima. To the best of our knowledge, we are the first to integrate unit propagation technique within LS to overcome local optima. Experiments based on a broad range of benchmarks from MaxSAT Evaluations, PBO competitions, the Mixed Integer Programming Library, and three real-life cases validate that our method significantly improves three state-of-the-art MaxSAT and PBO local search solvers.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases PBO, Partial MaxSAT, LS, CDCL

Digital Object Identifier 10.4230/LIPIcs.CP.2024.6

Funding *Pinyan Lu*: National Key R&D Program of China (2023YFA1009500).

1 Introduction

The Maximum Satisfiability (MaxSAT) and Pseudo-Boolean Optimization (PBO) are two fundamental and important constraint optimization problems. The Maximum Satisfiability problem (MaxSAT) is the optimization version of the Satisfiability problem (SAT). In general MaxSAT problems, clauses are divided into hard and soft clauses, and each soft clause has an associated weight. The goal of MaxSAT is to find an assignment that satisfies all hard clauses and maximizes the total weight of satisfied soft clauses. PBO consists of a set of pseudo-Boolean constraints and an objective function aiming to find a solution that satisfies all pseudo-Boolean constraints while minimizing the objective function. With the continuous improvements, MaxSAT and PBO solvers have broad applications in real-world problems from operations research, economics, manufacturing, etc. [19, 35, 37].

[1] X. Chen and Z. Lei - The authors are considered to have equal contributions
[2] Corresponding author.

Existing practical algorithms for MaxSAT and PBO can be classified into two categories: complete and incomplete methods. Local search is one of the most important incomplete methods and has been shown to be effective for solving many combinatorial optimization problems. Many advanced techniques have been proposed to enhance the performance of local search algorithms for MaxSAT, such as variable selection heuristics [8], clause weighting [11, 26], and Multi-armed Bandit soft clause selection [38]. Recently, local search algorithms for PBO have also achieved breakthroughs by using well-designed scoring functions for variable selection [12, 27]. Generally, local search algorithms converge quickly, enabling them to find high-quality feasible solutions within a reasonable time. As a result, they are widely employed for solving large instances, including real-world applications. However, local search algorithms often get stuck in local optima easily due to their limited reasoning ability and lack of global information.

On the contrary, complete algorithms adopt techniques that utilize global information and powerful reasoning, such as unit propagation (UP) and conflict-driven clause learning (CDCL) [34], to seek optimal solutions. Core-guided algorithms and branch-and-bound algorithms are two commonly used complete methods for solving PBO [10, 16, 17, 18, 33, 36] and MaxSAT [1, 2, 5, 6, 14, 15, 20, 23, 28, 31, 32] problems, where UP and CDCL play critical roles in improving the performance of these solvers. However, these methods are typically too time-consuming to solve some large-scale industrial instances.

Numerous research studies have been conducted to combine incomplete and complete methods to develop effective solution approaches for the constraint optimization problems. Some of these works utilize CDCL as the main solver, while LS is invoked to provide a search heuristic [13, 22, 30], or to perform deep search at a branching node [3, 9, 24, 28]. Other approaches use local search as the primary solver, with CDCL called upon for preprocessing [21, 25, 29], building initial assignments [7], or solving sub-problems as black boxes [4].

In sharp contrast to these combined methods, our work proposes a new search framework, in which local search acts as the main solver, and unit propagation techniques are used to help local search algorithms escape from local optima. Specifically, we first design a mechanism to detect when the local search algorithm is trapped in a local optimum. Upon detection, unit propagation is invoked to change the current assignment. Since binary constraints are common in MaxSAT and PBO instances, unit propagation will guide the solver into boarder search spaces, thereby offer more chance to jump out of the local optimum. Finally, we propose an acceptance criterion to determine whether to accept the propagated solution.

We have applied our new methods to improve the state-of-the-art MaxSAT solvers SATLike and NuWLS, as well as PBO solvers LS-PBO and NuPBO. Validation was conducted across it in MaxSAT Evaluations, PBO competitions, the Mixed Integer Programming Library, and three real-life cases. Experimental results demonstrate that the combination of UP technology significantly enhances the performance of local search algorithms.

2 Preliminary

2.1 Preliminaries Definitions and Notations

Given a set of n Boolean variables $x_1, x_2, \ldots, x_n$, a *literal* l_i is either a variable x_i (which is called a positive literal) or its negation $\overline{x_i} = 1 - x_i$ (which is called a negative literal). An *assignment* α is a mapping that assigns each variable a value (0 or 1).

A *clause* C_i of length k_i is a disjunction of k_i literals (i.e., $C_i = l_{i1} \vee l_{i2} \vee \cdots \vee l_{ik_i}$). A conjunctive normal form (CNF) formula $F = C_1 \wedge C_2 \wedge C_3 \cdots \wedge C_m$ is a conjunction of clauses. Given an assignment α, a clause is satisfied by α if it contains at least one true literal, and is falsified otherwise.

The Partial MaxSAT (PMS) problem is defined on a CNF formula, in which some clauses are designated as hard clauses and the reminder as soft. The objective is to find an assignment that satisfies all hard clauses and maximizes the total number of satisfied soft clauses. In Weighted PMS (WPMS), each soft clause is assigned a positive integer weight, and the goal is to satisfy all hard clauses while maximizing the total weight of satisfied soft clauses.

The Pseudo-Boolean Optimization (PBO) problem consists of a set of Pseudo-Boolean (PB) constraints and an objective function. A normalized Pseudo-Boolean constraint is represented as $\hat{C}_i : \sum_j a_{ij} \cdot l_{ij} \geq d_i$, where $a_{ij}, d_i \in \mathbb{Z}^+$ and l_{ij} are literals. The objective function is in the form $\sum_j c_{oj} \cdot l_{oj}$. The goal of PBO is to find an assignment that satisfies all PB constraints while minimizing the objective function.

Partial MaxSAT (PMS) can be considered as a specialization of PBO. Therefore, Partial MaxSAT instances can be readily encoded into PBO instances. Given a PMS instance with a set of hard clauses $H = \{C_1 \wedge C_2 \wedge \cdots \wedge C_m\}$, where $C_i = l_{i1} \vee l_{i2} \vee \cdots \vee l_{ik_i}$ and a set of soft clauses $S = \{S_1 \wedge S_2 \wedge \cdots \wedge S_n\}$, where $S_j = (w_j, l_{j1} \vee l_{j2} \vee \cdots \vee l_{jk_j})$ and w_j is the weight of the soft clause, the equivalent PBO format is constructed by transforming C_i's into constraints $\hat{C}_i : l_{i1} + l_{i2} + \cdots + l_{ik_i} \geq 1$, and S_j's into $\hat{C}_j : l_{j1} + l_{j2} + \cdots + l_{jk_j} + y_j \geq 1$, where y_j are auxiliary variables representing satisfaction of soft clauses. The objective function is $\sum_j w_j y_j$, which is weighted sum of auxiliary variables. Thus techniques used in PBO can be effectively adapted to PMS instances. For clarity and consistency, thorough the reminder of this paper, we discuss these concepts in the field of PBO.

2.2 Local Search and Unit Propagation

As shown in Algorithm 1, a local search solver maintains a complete assignment and keeps track of the best solution found during the search. In each step, the local search algorithm modifies the complete assignment locally (i.e., flips the value of a variable) to find a better solution and it returns the best solution when the termination condition is reached (lines 3-16).

State-of-the-art LS solvers often employ clause-weighting techniques. A dynamic weight is attached to each constraint to indicate its importance or difficulty. If an assignment α fails to satisfy a constraint $C_i : \sum_j a_{ij} l_{ij} \geq d_i$, the violation of C_i is defined as $\mathit{Violation}(C_i) = d_i - (\sum_j a_{ij} l_{ij})|_\alpha$, and the violation is 0 otherwise. In this way, the quality of an assignment can be evaluated by *Punishment*, which is defined as the weighted sum of *Violation* of all constraints. Specially, the objective function is treated as a never-satisfied constraint $C_o : \sum_j c_{oj} l_{oj} \geq M$ (where M is sufficiently large). In this way the objective is also counted into *Punishment*. The change in *Punishment* before and after a variable flip is then defined as the *score* of the corresponding variable.

In each step, if there exists a variable meeting the greedy heuristics, it is selected and flipped to decrease the Punishment(lines 8-9). If no such variable exists, or in other words the search get stuck, some local-optimum-escaping heuristics will be applied (lines 11-13). Typically, these heuristics involve increasing the weights of unsatisfied hard constraints and performing a random flip.

Unit propagation is the deduction of assignments over constraints. In the scope of MaxSAT, a *unit* clause is a clause containing only one literal. Generalizing this concept, a *unit* PB constraint is a constraint that cannot be satisfied unless one certain literal is satisfied. Similarly, a constraint is *binary*, if it becomes unit after one literal is falsified.

▶ **Example 1.** $2x_1 \geq 1$ and $3x_1 + x_2 + x_3 \geq 4$ are unit constraints, since they are unsatisfied unless $x_1 = 1$.

Algorithm 1 Typical Local Search Algorithm.

Input: A given instance F, *cutoff*
Output: An assignment α of F and its cost

α := an initial complete assignment.
α^* := $\emptyset$.
while elapsed time $<$ *cutoff* **do**
  if $cost[\alpha] < cost[\alpha^*]$ **then**
    $\alpha^* := \alpha$. ▷ update best solution
  end if
  if meet greedy heuristics **then**
    v := a variable is picked accordingly.
    $\alpha := \alpha$ with v flipped. ▷ greedy flips
  else
    update clause/constraint weights. ▷ weight adjustments
    v := a variable is picked according to local-optima-escaping heuristics.
    $\alpha := \alpha$ with v flipped. ▷ random flips
  end if
end while
Return $(\alpha^*, cost^*)$.

▶ **Example 2.** $2x_1 + x_2 + x_3 \geq 2$ is a binary constraint, since fixing $\overline{x_1} = 1$ leads to a unit constraint implying $x_2 = x_3 = 1$.

Generalized unit propagation works as follows: First, the algorithm assumes a literal l to be true, or in other words assigns a variable v a value of 0 or 1. Then, the algorithm accesses all the constraints that involve v and substitutes v with its value. If any constraints become unit, the value of another variable can be derived, and the derived variables will be assigned. The algorithm applies this rule iteratively until no new variable assignments can be derived or a conflict occurs, which means there exist some constraints that cannot be satisfied under current assignment. If no conflict occurs, the derived literals form the *Implication* of l, denoted as Imply(l).

▶ **Example 3.** Suppose there is a constraint $x_1 + x_2 + x_3 \geq 2$, then $\{x_2, x_3\} \subset \text{Imply}(\overline{x_1})$.

3 Main contribution

In this section, we present in detail how Systematic Search works and cooperates with local search. While we focus on PBO problems in this section, the method can be easily extended to PMS problems. The source code will be available at `https://github.com/SystematicSearch`.

3.1 Deep Cooperation of Local Search and Unit Propagation

The main drawback of local search is that it converges to a sub-optimal solution quickly, but the subsequent improvement is exceedingly challenging, because it easily gets trapped in a local optimum. We ran NuWLS, one of the state-of-the-art local search solver for PMS, for 300 seconds to solve the MSE benchmarks, and summarized the scores at different cutoff times in Figure 1 (benchmark and scoring will be introduced in Section 4). As shown in

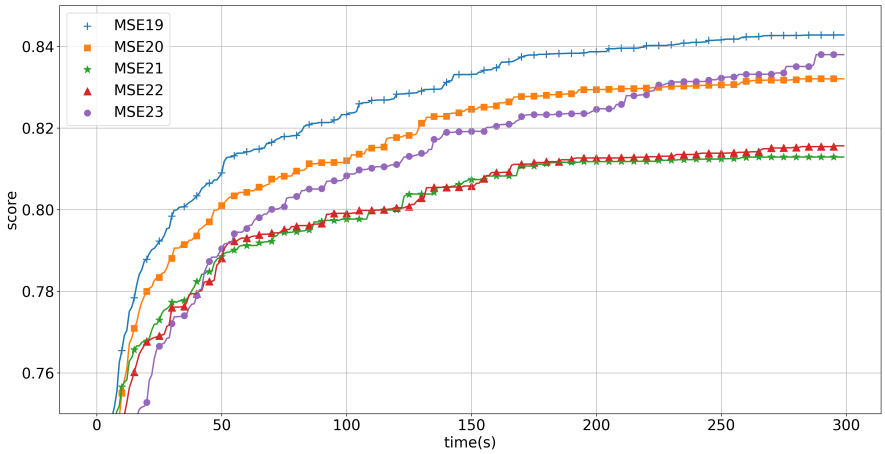

Figure 1 Baseline behavior: scores at different cutoff time of NuWLS.

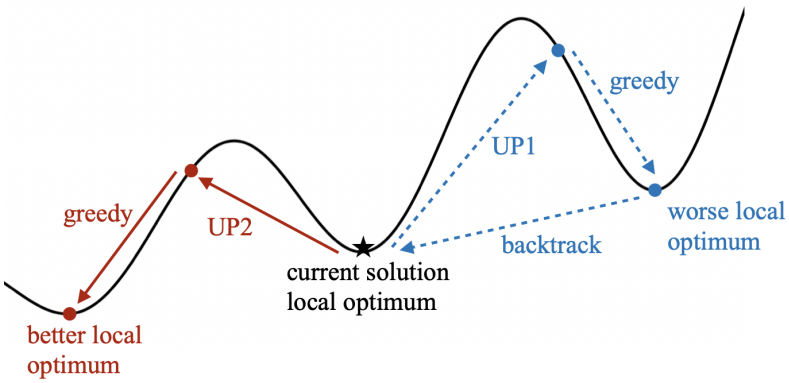

Figure 2 The horizontal direction represents the search direction, and the vertical direction indicates the *Punishment* of each assignment. Moving in the curve simulates the flips of local search.

the figure, the score rushes to 90% of its peak value in less than 10 seconds, but the last 100 seconds contribute only 0.01 to the score. Therefore, our motivation is to spend these seconds on addressing the trapping problem effectively.

To address this problem, we designed a new mechanism through deep cooperation of local search and unit propagation. As illustrated in Figure 2, the general idea of our approach is that, whenever the algorithm get stuck in a local optimum, a variable is picked and flipped together with the unit propagation procedure of this variable. Then, a greedy heuristic procedure is executed until the algorithm reaches another local optimum. If the new optimum is worse than the original one, we backtrack to the original assignment (the dashed line route). Otherwise, if the *Punishment* is reduced, we accept the new assignment (the solid line route). The modified LS framework involves Systematic Search if the original LS solvers get stuck or fail to improve the objectives, as described in Algorithm 2. Theoretically, this mechanism can be extended to all kinds of local search algorithms as long as a proper stuck-detection mechanism is designed.

Algorithm 2 Local Search Algorithm with Systematic Search.

Input: A given instance F, *cutoff*
Output: An assignment α of F and its cost

1: α := an initial complete assignment.
2: α^* := $\emptyset$.
3: **while** elapsed time $<$ *cutoff* **do**
4: **if** $cost[\alpha] < cost[\alpha^*]$ **then**
5: $\alpha^* := \alpha$. ▷ update best solution
6: **end if**
7: **if** meet Systematic Search invocation criteria **then**
8: *candidates* := PickCandidates(). ▷ see Section 3.3
9: **for all** $v \in$ *candidates* **do**
10: α' := FlipUP(v, α). ▷ see Section 3.4
11: **if** new assignment satisfy accept criteria **then**
12: $\alpha := \alpha'$ and **Break**. ▷ see Section 3.5
13: **end if**
14: **end for**
15: **else**
16: α : is modified by the original heuristic of the given algorithm.
17: **end if**
18: **end while**
19: **Return** $(\alpha^*,\ cost^*)$.

Our new local search scheme is illustrated in Algorithm 2 (lines 7-14). When the algorithm meets the systematic search invocation criteria (more details can be found in Section 3.2), which means it gets stuck in a local optimum, the algorithm will run the systematic search. The first step is to pick a candidate starting variable (line 8). Then the algorithm flips it, considers its assignment to be temporarily fixed, and performs the unit propagation procedure for this variable (line 10). Finally the new assignment will be evaluated. If the assignment is improved, we accept it and break (line 12). Otherwise, we backtrack to the last local optimum and try the next candidate. A thorough description of candidate selection, unit propagation, and acceptance criteria will be provided in the rest of this section.

3.2 Invocation of Hybrid Local Search

The hybrid method is involved when the best objective has not been updated for a long time, indicating that the original algorithm could be stuck. Intuitively, there are two favorable situations for invocation: when stuck in a local optimum, or right before restarting.

In the first situation, our new method is called every k times the algorithm reaches a local optimum, where k is initialized to a relatively small integer k_{init}. After each call, if the acceptance criteria are satisfied, indicating the method is effective, k will be reset to k_{init}. If all candidates fail, k will be doubled until it reaches a preset upper bound k_{max}. Theoretically, unit propagation does not need to be performed at every single local optimum for the following reasons. Firstly, since backtracking means flipping some variables twice without actual changes, frequent calls will cause a notable decrease of total valid flips, and may narrow the overall search space in the end. This effect will be discussed in Section 4. Secondly, updating weights plays a crucial role in balancing finding feasible solutions and reducing the objective cost, so we still need a number of weight updates and random flips (lines 11-13 in Algorithm 1). Finally, random flips is still one way to escape from some local optima. By keeping both methods, we can combine their strengths.

Algorithm 3 PickCandidates.

Input: current local optimum assignment
Output: a set of variables
Params: #unsat constraints chosen n_1, size of candidate n_2.

1: $candidates := \emptyset$.
2: **if** current assignment is infeasible **then**
3: **for** c: up to n_1 random unsatisfied constraint **do**
4: **for** l: all falsified literal in c **do**
5: $x :=$ the variable of l.
6: $candidates := candidates \cup \{x\}$.
7: **end for**
8: **end for**
9: **else**
10: **for** l: up to n_1 falsified literal in objective **do**
11: $x :=$ the variable of l.
12: $candidates := candidates \cup \{x\}$.
13: **end for**
14: **end if**
15: $candidates :=$ top n_2 elements with highest scores in $candidates$.
16: **Return** $candidates$.

The invocation in the second situation is based on the observation that feasible solutions are not uniformly distributed in the domain but rather cluster together. It is highly possible that a better solution lies near the best-found solution $\hat{\alpha}$, but the search might have chosen a different direction there. In this case, a deep search around the $\hat{\alpha}$ helps to improve the performance. To revisit that neighborhood, we flip to $\hat{\alpha}$, then call unit propagation to choose a search direction to leave $\hat{\alpha}$, followed by *final_steps* steps of the original search method. The restart will be delayed until all attempts are tried.

3.3 Picking Candidate Starting Variables

Our hybrid method begins with flipping one variable, followed by other propagated flips, so the correctness of the first flip is important. Similar to local-optimum-escaping heuristics in LS solvers, we filter some variables from unsatisfied constraints or falsified objective literals, and rank them by *score* to form a candidate set. As shown in Algorithm 3, if the local optimum is infeasible, (lines 2-8), at least one of falsified variable in each unsatisfied constraint needs to be flipped, so **PickCandidates** will visit falsified constraints, caching the falsified literals in *candidates* without duplication. Else, when the local optimum is feasible (lines 9-13), to improve the objective, at least one of the falsified literals in the objective function must be satisfied, so **PickCandidates** randomly picks among them. For the sake of efficiency, **PickCandidates** visits at most n_1 constraints and returns n_2 variables.

3.4 Flipping Based on Unit Propagation

It is challenging to obtain propagated information within a very tight time limit. To achieve efficient unit propagation, we propose a lightweight approach. Suppose we query for $UPList(x_0)$. Instead of calling a complete unit propagation over the formula, the algorithm checks every constraint containing the literal $\overline{x}_0$. For each of these constraints, the algorithm

Algorithm 4 FlipUP.

Input: a literal l, current assignment α
Output: local optimum assignment

```
1: U := UPList(l̄).
2: α := α with flip l → l̄.                          ▷ flip the candidate first
3: for all l′ ∈ U do
4:    if current assignments falsify l′ then
5:       α := α with flip l′ → l̄′.                   ▷ to be consist with UPList
6:    end if
7: end for
8: while meet original greedy heuristics do
9:    v := a variable is picked accordingly.
10:   flip(v).                                       ▷ follow the original greedy scheme
11: end while
12: Return α.
```

verifies if satisfying x_0 turns the constraint into a unit constraint. If so, we record the propagated literals in $UPList(x_0)$. Additionally, if time permits, we will run a breadth-first search on the propagated variables to gather more information. Sometimes a conflict occurs, such as when l and $\bar{l}$ appear simultaneously, or $\overline{x_0} \in UPList(x_0)$. These cases indicate that x_0 is an infeasible assignment, so the algorithm returns $UPList(x_0) = \{\overline{x_0}\}$ to prompt not to flip $\overline{x_0} \to x_0$. Since the implications of literals are unrelated to current assignments, *UPList* are cached to avoid redundant calculations.

▶ **Example 4.** Suppose a formula $\overline{x}_1 + x_2 \geq 1$, $\overline{x}_1 + x_3 \geq 1$, $\overline{x}_2 + \overline{x}_3 + x_4 \geq 1$, $\text{Imply}(x_1) = \{x_2, x_3, x_4\}$ but $UPList(x_1) = \{x_2, x_3\}$, or $UPList(x_2) \cup UPList(x_3)$ after a breadth first search.

The implementation of unit propagation is shown in Algorithm 4. First, the algorithm queries the *UPList* and flips until all literals in the *UPList* are satisfied (lines 3-7). Then, the search follows the original greedy heuristics and flips until a new local optimum is reached (lines 8-11). Consequently, we extend a single flip into a multi-step search action, which will strengthen the ability to escape from local optima.

The trick of getting the *UPList* works for several reasons. First of all, as an incomplete solver, our *UPList* is not obligated to be strictly complete and can therefore save time. Additionally, there will be greedy search steps after the propagation, providing another opportunity to flip those omitted variables. Finally, binary clauses are common in PMS and PBO instances, so the size of the derived *UPList* is large enough to move the search out of the local optimum, fulfilling our purpose.

3.5 Acceptance Criteria

In the previous text, we discussed that the motivation of our new method is to jump to another local optimum. If a correct candidate variable is chosen, we are supposed to get closer to the global optimum. However, when the candidate is a mistake, we ought to stop searching in that direction, go back, and try the next candidate (line 9 in Algorithm 2). We measure each assignment by *Punishment*, which is consistent with the measure of greedy search and the definition of a local optimum. If the *Punishment* decreases, we have successfully escaped the local optimum, so no more candidates need to be tried. Otherwise, our method ends

up in a worse local optimum, the new assignment is discarded by flipping changed variables again, followed by trying the next candidate. Finally, if all candidates fail to lead the search to a better local optimum, it is possible that unit propagation does not work in this situation, and the original local-optima-escaping heuristics will take place. In the next section, we will exhibit the proportion of success and failure of our method in the experiments.

4 Experiment Results

In this section, we implement our hybrid method on four state-of-the-art solvers for PMS and PBO problems, listed as follows:

- **LS-PBO**: The state-of-the-art local search PBO solver, proposed by [27].
- **NuPBO**: A recent PBO solver based on LS-PBO but using different scoring and weighting schemes, proposed by [12].
- **SATLike3.0**: The state-of-the-art local search MaxSAT solver and competition winner, proposed by [7].
- **NuWLS**: A recent MaxSAT solver based on SATLike3.0 but using different scoring and weighting schemes, proposed by [11].

We validate our approach on a wide range of benchmarks, described as follows:

- **CRAFTED**[3]: The crafted combinatorial problem set.
- **MIPLIB**[4]: A set of 0-1 integer linear programming problems.
- **PB16**[5]: The OPT-SMALLINT-LIN benchmark from the latest 2016 pseudo-Boolean competition.
- **Industry**[6]: A combination of some real-world problems, including the seating arrangement problem (SAP), the wireless sensor network optimization problem (WSNO), and the minimum-width confidence band problem (MWCB) [27].
- **MSE19-MSE23**[7]: The benchmarks used in the MaxSAT Evaluations in the last five years.

4.1 Experiment Settings

All competitor solvers are implemented in C++ and compiled with `g++` with the make option `-O3`. All the experiments are run on a workstation with an Intel Xeon Platinum 8380H CPU @ 2.90GHz. Each instance is solved once with a cutoff time of 300 seconds. The parameters of the planted local search solvers are kept the same as those used in [7, 11, 12, 27]. The newly added parameters used in our experiments are $n_2 = 5, k_{init} = 10, k_{max} = 2560$, and $final\_steps = 10000$. We will discuss the sensitivity of results to these parameters and random seeds in Section 4.4.

We conclude the results in two dimensions, the number of winning instances and the average score. Each experiment compares the original algorithm with its unit-propagation-implemented version. For each instance, if the results are different, #win for the superior algorithm is incremented by 1, and #lose for the inferior algorithm is also incremented by 1. The score for each result follows the method used in MSE competitions. Specifically, within

[3] https://zenodo.org/record/4036016
[4] https://zenodo.org/record/3870965
[5] http://www.cril.univ-artois.fr/PB16/bench/PB16-used.tar
[6] https://lcs.ios.ac.cn/~caisw/Resource/LS-PBO/benchmark/
[7] https://maxsat-evaluations.github.io/2023/benchmarks.html

each experiment set, let v^* denote the best objective found by all competing algorithms. A solver returning objective v receives a score of $(1+v^*)/(1+v)$. This score is 1 if $v = v^*$, less than 1 if it is not the best result, and equals 0 if no feasible solution is found. Given that v^* is non-negative in the context of MaxSAT and PBO problems, scores range between 0 and 1. We report the average score across all benchmarks in this section.

To analyze the performance of our method, we also record several other indices. *accept(%)* represents the acceptance rate, calculated as the percentage of accepted local optima over the total number of candidate attempts. *dist* denotes the average distance, or the Hamming distance between two local optima before and after an accepted *FlipUP* action. *step(%)* indicates the average number of valid step, which is the sum of greedy steps, random flips, and the number of flips in accepted *FlipUP* actions, but excludes rejected propagations and backtrack flips.

4.2 Experiment Results on MaxSAT Benchmarks

Results on SatLike3.0 and NuWLS are shown in Table 1 and 2 respectively. Both solvers perform significantly better with the cooperation of unit propagation, evidenced by the increase in the number of winning instances and the average scores across these benchmarks. The progress of scores over time is depicted in Figure 3. For MSE19-22, the score increase of original NuWLS slows down at around 0.72, whereas our approach delays this slowdown until around 0.76. However, in MSE23, a distinct pattern emerges where efficiency becomes more crucial. The distance between the solid curve and the dotted curve was narrowed to nearly zero twice (around 50s and 200s) and then widened again (around 130s and 290s). We attribute this pattern to the fact that some improvements at these timestamps are delayed because the unit-propagation-implemented iterations are slower than the original ones.

Under the acceptance criteria, approximately 30 to 40 percent of propagation attempts are accepted, resulting in the search jumps from current local minima to around 6 flips away. In contrast, the rejected attempts are purely wastes of time. The average time cost of our method, including calculating *UPList* and performing backtracking, amounts to approximately 15 to 20 percent of the cutoff time. The results depicted in Figure 1 suggest that this amount of time overhead has limited impact on the average score, but there are still some instances to be sensitive to the effective running time. Particularly in MSE23, the score curve showed steep improvements after 200 seconds, indicating that the solver finally finds a feasible solution and turns the score from 0 to 1. In these cases, the unit propagation integrated version ends up with no feasible solution, which explains for the score decrease in MSE23 in Table 1.

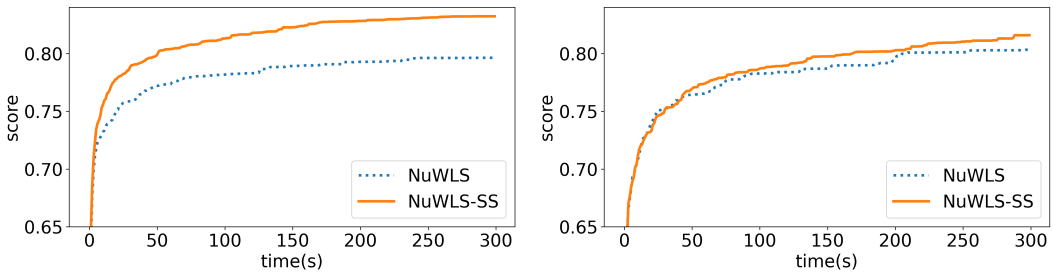

Figure 3 Comparison of the score in different cutoff time, by NuWLS. The left and right plots show MSE19 and MSE 23 respectively. The pattern of MSE20, MSE21 and MSE22 are so similar to MSE19, so we omit these plots here to be concise.

Table 1 Experiment results of SatLike3.0.

benchmark	#inst.	#win.	#lose.	$satlike_{ss}$	$satlike$	$accept(\%)$	$dist$	$step(\%)$
MSE19	299	**72**	13	**0.7108**	0.6911	39	7.6	79
MSE20	262	**59**	13	**0.6923**	0.6845	41	8.4	80
MSE21	155	**30**	10	**0.6155**	0.6065	40	5.0	77
MSE22	179	**39**	9	**0.6637**	0.6459	42	4.9	82
MSE23	179	**61**	16	0.6193	**0.6257**	33	6.0	80

Table 2 Experiment results of NuWLS.

benchmark	#inst.	#win.	#lose.	$nuwls_{ss}$	$nuwls$	$accept(\%)$	$dist$	$step(\%)$
MSE19	299	**74**	38	**0.8322**	0.7963	28	4.5	85
MSE20	262	**75**	30	**0.8219**	0.7882	30	5.0	86
MSE21	155	**34**	19	**0.7991**	0.7723	27	4.7	84
MSE22	179	**46**	25	**0.8044**	0.7856	28	4.0	86
MSE23	179	**61**	30	**0.8158**	0.8032	25	5.3	86

4.3 Experiment Results on PBO benchmarks

We have also applied our methods in PBO solvers LS-PBO and NuPBO, and summarized the results in Table 3 and 4. The conclusion is consistent with that in the previous subsection: in CRAFTED, MIPLIB, and PB16 benchmarks, both the number of winning cases and the score are significantly improved. The only deterioration falls in the score of Industry problem set by NuPBO, which can be explained by the $step(\%)$ metric in Table 4. Given that the Industry set comprises larger-scale problems compared to other benchmarks, the original solver usually cannot converge in 300 seconds, in other words the solution quality highly depends on the number of flips. However, our new method achieves only performed 69% valid flips compared to the original solver. As a result, despite unit propagation demonstrating effectiveness by achieving more winning cases, the average score drops slightly.

4.4 Sensitivity analysis

Our algorithm shows stability across a wide range of parameter values. In each of the following experiments, we vary one parameter while keeping the rest as specified in Section 4.1. For each solver, we conduct 27 experiments (1 default, 6 for $k_{\max}$, 4 for $k_{\min}$, 3 for $final\_steps$, 3 for n_2, and 10 for random seeds) on all benchmarks in our study. We compute the scores in comparison with the best value among these 27 outputs. For conciseness, we present the average score of all benchmarks for each experiment in Table 5.

The first part of Table 5 shows the average scores and standard errors of experiments with 10 different random seeds. The statistics demonstrate that the algorithm is robust against randomness.

$k_{\max}$ and $k_{\min}$ control the frequency of UP calls. The results of varying these parameters are shown in the second and third parts of Table 5. $k_{\min}$ corresponds to the highest frequency of UP calls. As discussed in Section 3.2, a smaller $k_{\min}$ results in a more significant UP effect but consumes more time. Conversely, a larger $k_{\min}$ reduces the frequency of UP calls, making performance closer to the original solver. Setting $k_{\min} = \infty$ means no UP is conducted, making the algorithm identical to the original.

$k_{\max}$ corresponds to the lowest frequency of UP calls. This parameter is set to reduce time waste when UP is ineffective during the search. We tested $k_{\max}$ from 10 ($= k_{\min}$) to 10240 (extremely low frequency). The performance is not sensitive to this parameter.

The results of varying the number of UP candidates n_2 are shown in the fourth part of Table 5. A larger n_2 increases the chances of finding a better UP but also increases the time cost. We chose a balanced value of 5 for our paper.

The results of varying $final\_steps$, the extra steps before restart (described in Section 3.2), are shown in the last part of Table 5. Setting $final\_steps = 0$ disables revisiting the best-found solution, resulting in less intensified search and relatively lower scores. There is not much difference when using other values.

4.5 Validation of Acceptance Criteria

In this part, we tested a variant that accepts all local optima after unit propagation. The results of the SATLike3.0-based experiment are shown in Table 6, while the other solvers support the same conclusion. Across all benchmarks, the acceptance rate is increased to around 80% because only conflicting flips or candidates with an empty *UPList* were rejected. As a result, approximately 10% more valid steps could be attempted before the cutoff time. However, the number of deteriorated instances is significantly outweighed the number of improved ones, and the score drops across four benchmarks at the same time. These results demonstrate the necessity of our acceptance criteria. Moreover, it is worth noting that the average propagated distance *dist* decreased compared to Table 2. As mentioned in the previous text, solutions often cluster closely together. If the algorithm has no enforcement to accept a better local optimum, it is allowed to visit these solutions in loops, which means it is stuck.

5 Related Works

There have been many applications in combining Local Search and CDCL solvers. In this section, we classify these works into two categories based on whether Local Search is the master solver, and discuss some noticeable works from each category.

The first category use CDCL as the main solver, with local search methods often viewed as a black box. In SAT problems, the solve ends if a feasible solution is found, prompting some hybrid SAT solvers to use LS on branch nodes to accelerate solving. [30] calls LS at every node in the CDCL search tree, and the solve succeeds if LS finds a solution at any nodes. [9] branches until the length of the partial assignment exceeds a threshold, then it is extended into a complete assignment and passes as the initial assignment to a LS solver. Some other works also use LS to estimate the priority of branch variables ([13]), or to obtain a upper bound of the optimization model ([28]).

The second category considers LS as the main body. [29] shows learnt clauses by CDCL will be beneficial to LS solving. [21] uses implication graph to discover variable equivalency, so the redundant variables will be substituted. The most related work is SatHys proposed in [3], where the CDCL solver maintains a partial assignment $\mathcal{I}_p$ while LS conducts a search with complete assignment $\mathcal{I}_c$. If LS is stuck, a literal l is added to $\mathcal{I}_c$ and propagated. If conflict occurs, CDCL learns a clause and backtracks. Finally $\mathcal{I}_c$ will be overwritten by $\mathcal{I}_p$. Our approach differs in two aspects. The major difference is that our unit propagation is based on one literal instead of on $\mathcal{I}_p$, which is a better adaption to optimization problems. We notice that in SatHys, if an incorrect decision is made and added in $\mathcal{I}_p$, it cannot be cancelled unless it is backtracked or restarted. It works in SAT problems because the

Table 3 Experiment results of LS-PBO.

benchmark	#inst.	#win.	#lose.	$lspbo_{ss}$	$lspbo$	$accept(\%)$	$dist$	$step(\%)$
CRAFTED	955	**38**	6	**0.9093**	0.9081	46	2.4	78
MIPLIB	291	**76**	36	**0.7260**	0.7042	32	5.4	88
PB16	1600	**346**	92	**0.7653**	0.7147	30	8.6	80
Industry	63	**45**	9	**0.9835**	0.9295	47	3.4	80

Table 4 Experiment results of NuPBO.

benchmark	#inst.	#win.	#lose.	$nupbo_{ss}$	$nupbo$	$accept(\%)$	$dist$	$step(\%)$
CRAFTED	955	**46**	6	**0.9171**	0.9168	38	2.8	79
MIPLIB	291	**95**	34	**0.8219**	0.7932	27	8.9	82
PB16	1600	**269**	88	**0.8293**	0.8283	24	20.3	81
Industry	63	**32**	20	0.9456	**0.9561**	47	3.7	69

Table 5 Performance of our hybrid algorithm under different parameter settings.

(seeds)	avg.score	std.err
LS-PBO	0.8011	0.0015
NuPBO	0.8379	0.0007
SATLike	0.6503	0.0044
NuWLS	0.7835	0.0030

$k_{\min}=$	5	10	20	40	80	∞
LS-PBO	0.8145	0.8010	0.7984	0.7858	0.7823	0.7762
NuPBO	0.8356	0.8386	0.8381	0.8376	0.8382	0.8360
SATLike	0.6527	0.6493	0.6491	0.6491	0.6436	0.6405
NuWLS	0.7914	0.7867	0.7784	0.7763	0.7733	0.7626

$k_{\max}=$	10	160	640	1280	2560	5120	10240
LS-PBO	0.8055	0.7997	0.7988	0.8049	0.8010	0.8017	0.8016
NuPBO	0.8326	0.8368	0.8353	0.8368	0.8386	0.8380	0.8370
SATLike	0.6459	0.6471	0.6493	0.6506	0.6496	0.6470	0.6471
NuWLS	0.7869	0.7823	0.7829	0.7861	0.7867	0.7902	0.7891

$final\_steps=$	0	1000	10000	100000
LS-PBO	0.7978	0.8021	0.8010	0.8029
NuPBO	0.8358	0.8370	0.8386	0.8380
Satlike	0.6449	0.6473	0.6493	0.6470
NuWLS	0.7891	0.7922	0.7867	0.7899

$n_2=$	1	5	10	15
LS-PBO	0.7770	0.8010	0.8131	0.8155
NuPBO	0.8373	0.8386	0.8374	0.8402
Satlike	0.6490	0.6493	0.6490	0.6560
NuWLS	0.7733	0.7867	0.7867	0.7890

Table 6 Experiment results of SATLike3.0, comparing normal version $satlike_{ss}$ with an all-accept variant $satlike_{any}$. The last three columns of statistics are from $satlike_{any}$.

benchmark	#inst.	#win.	#lose.	$satlike_{ss}$	$satlike_{any}$	$accept(\%)$	$dist$	$step(\%)$
MSE19	299	**65**	17	**0.7132**	0.6962	83	6.0	106
MSE20	262	**54**	17	**0.6920**	0.6831	85	5.4	112
MSE21	155	**28**	11	**0.6041**	0.5761	86	3.2	113
MSE22	179	**43**	13	**0.6508**	0.6430	85	3.6	112
MSE23	179	**48**	22	0.6193	**0.6210**	79	3.5	104

cancelling situation (i.e. conflict occurs) is consistent with the target to find a feasible assignment. But in optimization problems, the target is to optimize an objective, and poor decisions (with respect to objective) may be kept in $\mathcal{I}_p$ and cause LS ending up in feasible but bad solutions. Also, the adaption of our work allows the tricks described in section 3.4 to promise a light and efficient way of unit propagation. Another novelty of our work lies in the backtrack mechanism, it gives LS the highest authority to judge propagated assignments with LS *Punishment*, and reject unfavorable CDCL moves.

6 Conclusion

We propose a novel search framework which embeds unit propagation into local search algorithms to help escaping local optima. Then, we introduce three innovative ideas to enhance this new search framework. The first one is a mechanism to determine when to invoke UP to change the current assignment obtained by the local search method, which helps the algorithm jump out of local optima. The second one is the *FlipUP* algorithm to perform unit propagation and heuristic search. Finally, we propose an acceptance mechanism to decide whether to accept the propagated solution. Experiments conducted on benchmarks from MaxSAT Evaluations, PBO competitions and realistic instances demonstrate that our method can bring significant improvement in terms of the number of winning instances and average scores across four state-of-the-art MaxSAT and PBO local search solvers. Consequently, we believe that the cooperation of local search and unit propagation techniques represents a promising research field which has great potential for exploration.

References

1 Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSAT algorithms. *Artif. Intell.*, 196:77–105, 2013.

2 Carlos Ansótegui and Joel Gabàs. WPM3: an (in)complete algorithm for weighted partial maxsat. *Artif. Intell.*, 250:37–57, 2017.

3 Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Sais. Boosting local search thanks to cdcl. In Christian G. Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings*, volume 6397 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2010.

4 Adrian Balint, Michael Henn, and Oliver Gableske. A novel approach to combine a sls-and a dpll-solver for the satisfiability problem. In *Theory and Applications of Satisfiability Testing-SAT 2009: 12th International Conference, SAT 2009, Swansea, UK, June 30-July 3, 2009. Proceedings 12*, pages 284–297. Springer, 2009.

5 Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete maxsat. In Louis-Martin Rousseau and Kostas Stergiou, editors, *Proceedings CPAIOR 2019,*, volume 11494, pages 39–56, 2019.

6 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–64, 2010.

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

8 Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial MaxSAT. In *Proceedings of AAAI 2014*, pages 2623–2629, 2014.

9 Shaowei Cai and Xindi Zhang. Deep cooperation of CDCL and local search for SAT (extended abstract). In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 5274–5278. ijcai.org, 2022.

10 Donald Chai and Andreas Kuehlmann. A fast pseudo-Boolean constraint solver. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(3):305–317, 2005.

11 Yi Chu, Shaowei Cai, and Chuan Luo. Nuwls: Improving local search for (weighted) partial maxsat by new weighting techniques. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 3915–3923. AAAI Press, 2023.

12 Yi Chu, Shaowei Cai, Chuan Luo, Zhendong Lei, and Cong Peng. Towards more efficient local search for pseudo-boolean optimization. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

13 James M Crawford. Solving satisfiability problems using a combination of systematic and local search. In *Second DIMACS Challenge: cliques, coloring, and satisfiability*. Citeseer, 1993.

14 Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proceedings of CP 2011*, pages 225–239, 2011.

15 Emir Demirovic and Peter J. Stuckey. Techniques inspired by local search for incomplete maxsat and the linear algorithm: Varying resolution and solution-guided search. In Thomas Schiex and Simon de Givry, editors, *Proceedings of CP 2019*, volume 11802, pages 177–194, 2019.

16 Jo Devriendt, Stephan Gocht, Emir Demirović, Peter Stuckey, and Jakob Nordström. Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In *AAAI 2021,Accepted*, 2021. URL: `http://www.csc.kth.se/~jakobn/research/CuttingToTheCore_AAAI.pdf`.

17 Jan Elffers, Jesús Giráldez-Cru, Jakob Nordström, and Marc Vinyals. Using combinatorial benchmarks to probe the reasoning power of pseudo-Boolean solvers. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proceedings of SAT 2018*, pages 75–93, 2018.

18 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.

19 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006.

20 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proceedings of SAT 2006*, pages 252–265, 2006.

21 Djamal Habet, Chu Min Li, Laure Devendeville, and Michel Vasquez. A hybrid approach for sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 172–184. Springer, 2002.

22 William S Havens and Bistra N Dilkina. A hybrid schema for systematic local search. In *Advances in Artificial Intelligence: 17th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2004, London, Ontario, Canada, May 17-19, 2004. Proceedings 17*, pages 248–260. Springer, 2004.

23 Saurabh Joshi, Prateek Kumar, Sukrut Rao, and Ruben Martins. Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *J. Satisf. Boolean Model. Comput.*, 11(1):73–97, 2019.

24 Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, 2002.

25 Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. Maxpre: an extended maxsat preprocessor. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 449–456. Springer, 2017.

26 Zhendong Lei and Shaowei Cai. Solving (weighted) partial maxsat by dynamic local search for SAT. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 1346–1352, 2018.

27 Zhendong Lei, Shaowei Cai, Chuan Luo, and Holger H. Hoos. Efficient local search for pseudo boolean optimization. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 332–348. Springer, 2021.

28 Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamal Habet, and Kun He. Combining clause learning and branch and bound for maxsat. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 38:1–38:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

29 Jan-Hendrik Lorenz and Florian Wörz. On the effect of learned clauses on stochastic local search. In *Theory and Applications of Satisfiability Testing–SAT 2020: 23rd International Conference, Alghero, Italy, July 3–10, 2020, Proceedings 23*, pages 89–106. Springer, 2020.

30 Bertrand Mazure, Lakhdar Sais, and Éric Grégoire. Boosting complete techniques thanks to local search methods. *Annals of mathematics and artificial intelligence*, 22:319–331, 1998.

31 Alexander Nadel. Anytime weighted maxsat with improved polarity selection and bit-vector optimization. In Clark W. Barrett and Jin Yang, editors, *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 Steven Prestwich. Randomised backtracking for linear pseudo-Boolean constraint problems. In *Proceedings of CPAIOR 2002*, pages 7–20, 2002.

34 João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiabilitysatlike. In Rob A. Rutenbar and Ralph H. J. M. Otten, editors, *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*, pages 220–227. IEEE Computer Society / ACM, 1996.

35 Robert Wille, Hongyan Zhang, and Rolf Drechsler. ATPG for reversible circuits using simulation, boolean satisfiability, and pseudo boolean optimization. In *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2011, 4-6 July 2011, Chennai, India*, pages 120–125. IEEE Computer Society, 2011.

36 Aolong Zha, Miyuki Koshimura, and Hiroshi Fujita. A hybrid encoding of pseudo-Boolean constraints into CNF. In *Proceedings of TAAI 2017*, pages 9–12. IEEE Computer Society, 2017.

37 Yuhang Zhang, Richard I. Hartley, John Mashford, and Stewart Burn. Superpixels via pseudo-boolean optimization. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc Van Gool, editors, *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 1387–1394. IEEE Computer Society, 2011.

38 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 Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 1901–1907. ijcai.org, 2022.