

# Farsighted Probabilistic Sampling: A General Strategy for Boosting Local Search MaxSAT Solvers

Jiongzhi Zheng<sup>1,2</sup>, Kun He<sup>1,2\*</sup>, Jianrong Zhou<sup>1,2</sup>

<sup>1</sup>School of Computer Science and Technology, Huazhong University of Science and Technology, China

<sup>2</sup>Hopcroft Center on Computing Science, Huazhong University of Science and Technology, China  
{jzzheng, brooklet60, jrzhou}@hust.edu.cn

## Abstract

Local search has been demonstrated as an efficient approach for two practical generalizations of the MaxSAT problem, namely Partial MaxSAT (PMS) and Weighted PMS (WPMS). In this work, we observe that most local search (W)PMS solvers usually flip a single variable per iteration. Such a mechanism may lead to relatively low-quality local optimal solutions, and may limit the diversity of search directions to escape from local optima. To address this issue, we propose a general strategy, called farsighted probabilistic sampling (FPS), to replace the single flipping mechanism so as to boost the local search (W)PMS algorithms. FPS considers the benefit of continuously flipping a pair of variables in order to find higher-quality local optimal solutions. Moreover, FPS proposes an effective approach to escape from local optima by preferring the best to flip among the best sampled single variable and the best sampled variable pair. Extensive experiments demonstrate that our proposed FPS strategy significantly improves the state-of-the-art (W)PMS solvers, and FPS has an excellent generalization capability to various local search MaxSAT solvers.

## 1 Introduction

Maximum Boolean Satisfiability (MaxSAT) is an optimization version of the famous SAT decision problem. Given a propositional formula in the Conjunctive Normal Form (CNF), MaxSAT aims to maximize the number of satisfied clauses. Partial MaxSAT (PMS) is a generalization of MaxSAT, whose clauses are divided into hard and soft. PMS aims to maximize the number of satisfied soft clauses while satisfying all the hard clauses. In a more generalized situation, each soft clause is associated with a positive weight. The resulting problem is called Weighted PMS (WPMS), which aims to maximize the total weight of satisfied soft clauses meanwhile satisfying all the hard clauses. Both PMS and WPMS, denoted as (W)PMS, have many practical applications, such as planning (Bonet, Francès, and Geffner 2019), timetabling (Demirovic and Musliu 2017), routing (Khadilkar 2022), group testing (Ciampiconi et al. 2020), *etc.*

Local search (Selman, Kautz, and Cohen 1993; Morris 1993; Cha et al. 1997) is a well-studied category of incom-

plete algorithms for MaxSAT. One of the most common frameworks of local search MaxSAT algorithms starts from an initial solution, and then flips a variable in each iteration to explore the solution space. Recently, many effective local search strategies have been proposed for (W)PMS, such as approaches for generating high-quality initial solutions (Cai, Luo, and Zhang 2017; Cai and Lei 2020; Zheng et al. 2022), variable selection strategies (Luo et al. 2017; Zheng et al. 2022), and clause weighting schemes (Cai et al. 2014; Luo et al. 2017; Lei and Cai 2018).

These state-of-the-art local search algorithms (Cai et al. 2014; Luo et al. 2017; Lei and Cai 2018; Cai and Lei 2020; Zheng et al. 2022) have made considerable achievements in solving the (W)PMS. However, they all follow a similar single flipping mechanism to explore the solution space. That is, they only flip a single variable in each iteration. Such a mechanism may cause the algorithm easily fall into a local optimum, *i.e.*, flipping any variable can not improve the current solution. Thus the quality of the local optimal solutions might not be good enough. Moreover, most of these algorithms escape from local optima by a simple random walk strategy (Cai et al. 2014; Lei and Cai 2018; Cai and Lei 2020), *i.e.*, satisfying a randomly selected falsified clause. The high-degree randomness of this strategy may make it hard for these algorithms to find a good direction for the subsequent search.

To handle these issues, we propose a general variable selection strategy, dubbed farsighted probabilistic sampling (FPS), to replace the single flipping mechanism in the local search (W)PMS solvers. FPS employs a two-level technique, that allows the algorithm to look-ahead and considers the benefit of continuously flipping a pair of variables, as well as a probabilistic sampling approach. First, when there exists no single flipping that can improve the current solution (*i.e.*, a local optimum of the single flipping mechanism reaches), FPS tries to look-ahead to find a pair of variables that flipping both can improve the current solution. In this way, the local optimal solutions can be further improved. Second, the two-level technique and the sampling strategy can provide more and better search directions to escape from the local optima. If FPS fails to improve the current solution by flipping a pair of variables, it will choose the best to flip among the best sampled single variable and the best sampled pair of variables.

\*Corresponding author.

In the literature, there exist related studies that apply the look-ahead or similar techniques for the SAT problem. Such as the look-ahead technique (Li and Huang 2005; Li, Wei, and Zhang 2007; Wei, Li, and Zhang 2008) and an approach using the second level score (Cai and Su 2013; Cai, Su, and Luo 2013). To evaluate the benefit of flipping a variable, these two approaches consider the immediate benefit of the flipping, as well as the future benefit that might be obtained after the flipping. However, they still prioritize the immediate benefit of flipping a single variable, and always perform a single flipping per iteration. For example, the look-ahead technique is used to select one among the two best single flipping variables (Li, Wei, and Zhang 2007; Wei, Li, and Zhang 2008) and the second level score is used only for breaking ties (Cai and Su 2013; Cai, Su, and Luo 2013). Thus the look-ahead and second level score techniques play a minor role in selecting the flipping variable. Intuitively, prioritizing the immediate benefit over the future benefit is short-sighted, and may lead to a poor search direction.

There are also some studies proposing multi-flipping local search operators for SAT (Mali and Lipen 2003) and MaxSAT (Reisch, Großmann, and Kliewer 2020). But they use an exhaustive method rather than probability sampling in the local search process. Thus they are less efficient when solving large instances. In contrast, our proposed FPS algorithm combines the advantages of look-ahead and probability sampling. Thus FPS is more effective and efficient for solving the (W)PMS.

To evaluate the performance of FPS, we apply FPS to some of the state-of-the-art local search (W)PMS algorithms, including BandMaxSAT (Zheng et al. 2022), SAT-Like3.0 (Cai and Lei 2020) (the newest extension of SAT-Like (Lei and Cai 2018)), CCEHC (Luo et al. 2017), and Dist (Cai et al. 2014). Details of how to apply FPS to these algorithms are described in Section 3.2. The results show that these algorithms can all be improved significantly by FPS, demonstrating its excellent generalization capability. Moreover, we make a further comparison with some of the state-of-the-art SAT-based (W)PMS solvers, including SATLike-c (Lei et al. 2021), TT-Open-WBO-Inc (Nadel 2019), and Loandra (Berg, Demirovic, and Stuckey 2019). We apply FPS to improve the local search component in SATLike-c, the resulting solver also outperforms these SAT-based (W)PMS solvers.

The main contributions of this work are as follows:

- We propose a general farsighted probabilistic sampling (FPS) strategy for boosting local search MaxSAT solvers. Extensive experiments demonstrate that FPS significantly improves the state-of-the-art local search (W)PMS solvers as well as one of the state-of-the-art SAT-based solvers, SATLike-c, which won three among all the four incomplete tracks in MaxSAT Evaluation 2021.
- FPS can improve the local optimal solutions of the single flipping mechanism by considering the benefit of continuously flipping a pair of variables. FPS escapes from local optima by preferring the best to flip among the best sampled single variable and the best sampled pair of variables, which can provide high-quality search directions

to escape from local optima.

- Our method suggests an efficient and effective way to apply the look-ahead strategy and the multiple flipping mechanism to boost local search MaxSAT solvers, showing great potential for these approaches for MaxSAT.

## 2 Preliminary

In this section, we first present formal definitions of the studied problems and some important concepts. Then we summarize the general framework of the state-of-the-art local search (W)PMS algorithms based on the single flipping mechanism, which can help understand how FPS can be applied to boost them (described in Section 3.2).

### 2.1 Definitions on Problems and Concepts

Given a set of Boolean variables  $\{x_1, \dots, x_n\}$ , a literal is either a variable  $x_i$  itself or its negation  $\neg x_i$ , a clause  $c$  is a disjunction of literals, and a conjunctive normal form (CNF) formula  $\mathcal{F}$  is a conjunction of clauses. A complete assignment  $A$  is a mapping that assigns to each variable either 1 (true) or 0 (false). A literal  $x_i$  (resp.  $\neg x_j$ ) is true if  $x_i = 1$  (resp.  $x_j = 0$ ). A clause is satisfied if it has at least one true literal, and falsified otherwise.

Given a CNF formula, SAT aims to determine whether there is an assignment that satisfies all the clauses in the formula, and MaxSAT aims to find an assignment that maximizes the number of satisfied clauses. The PMS, for which the clauses are divided into hard and soft, aims to find an assignment that satisfies all the hard clauses and satisfies as many soft clauses as possible. The WPMS, for which the soft clauses are associated with positive weights, aims to find an assignment that satisfies all hard clauses and maximizes the total weight of the satisfied soft clauses.

For a (W)PMS instance  $\mathcal{F}$ , an assignment  $A$  is regarded as feasible if it satisfies all the hard clauses in  $\mathcal{F}$ , and the cost of a feasible assignment  $A$ , denoted by  $cost(A)$ , is defined to be the number (or total weight) of the falsified soft clauses. For convenience, the cost of any infeasible assignment is set to  $+\infty$ . The flipping operator in local search algorithms for MaxSAT on a variable is to change its Boolean value. Recent effective local search (W)PMS algorithms use the clause weighting strategy that associates dynamic weights to both hard and soft clauses to guide the search direction. Some of them, such as BandMaxSAT and SATLike(3.0), use a single scoring function  $score(x)$  to represent the increase of the total dynamic weight of the satisfied hard and soft clauses caused by flipping  $x$ . Others such as CCEHC and Dist use  $hscore(x)$  (resp.  $sscore(x)$ ) to represent the increase of the total dynamic weight of the satisfied hard (resp. soft) clauses caused by flipping  $x$ .

### 2.2 General Single Flipping Local Search

The general flow of local search (W)PMS algorithms based on the single flipping mechanism is shown in Algorithm 1. We mainly focus on the variable selection process in each iteration (lines 5-9), which can be divided into two cases. In the first case (lines 5-6), the current assignment  $A$  is not a local optimum, i.e.,  $GoodVars \neq \emptyset$ . Different algorithms de-

---

**Algorithm 1:** General Single Flipping Local Search

---

**Input:** A (W)PMS instance  $\mathcal{F}$ , cut-off time *cutoff***Output:** A feasible assignment  $A$  of  $\mathcal{F}$ , or no feasible assignment found

```
1  $A :=$  an initial assignment;  $A^* := A$ ;  
2 while running time < cutoff do  
3   if  $A$  is feasible &  $\text{cost}(A) < \text{cost}(A^*)$  then  
4      $A^* := A$ ;  
5   if  $\text{GoodVars} \neq \emptyset$  then  
6      $v :=$  a variable selected from  $\text{GoodVars}$ ;  
7   else  
8      $c :=$  a selected falsified clause;  
9      $v :=$  a variable selected from  $c$ ;  
10   $A := A$  with  $v$  flipped;  
11 if  $A^*$  is feasible then return  $A^*$ ;  
12 else return no feasible assignment found;
```

---

fine different *GoodVars*. For example, BandMaxSAT and SATLike(3.0) define *GoodVars* as  $\{x | \text{score}(x) > 0\}$ , Dist and CCEHC define *GoodVars* as  $\{x | \text{hscore}(x) > 0 \vee (\text{hscore}(x) = 0 \wedge \text{sscore}(x) > 0)\}$ . CCEHC further uses the configuration checking strategy (Cai, Su, and Sattar 2011) to refine *GoodVars*. In general, flipping a variable in *GoodVars* results in a better solution than the current one. The algorithms use a greedy (Luo et al. 2017) or sampling strategy (Lei and Cai 2018; Cai and Lei 2020) to select a variable to be flipped in this case. Anyhow, the variable selection strategies of these algorithms in this case are reasonable, since the current solution can always be improved.

In the second case (lines 7-9), the algorithms fall into a local optimum. A common strategy is to select a falsified clause  $c$  (usually with a bias to hard clauses) first (line 8), and then select a variable  $v$  from  $c$  (line 9). The Dist, CCEHC, and SATLike(3.0) algorithms use the simple random walk strategy to select the clause to be satisfied (*i.e.*,  $c$ ) randomly. The recently proposed BandMaxSAT algorithm uses its multi-armed bandit model to select the clause  $c$  smartly, which can help the algorithm find a good search direction. After determining  $c$ , the algorithms usually select  $v$  greedily according to their scoring functions. Besides, before using the random walk strategy, CCEHC tries to greedily (according to its scoring functions) select the variable to be flipped that satisfies the configuration checking condition (Luo et al. 2017) in all the falsified clauses.

### 3 Methodology

We propose a general farsighted probabilistic sampling (FPS) variable selection strategy to replace or improve the single flipping mechanism that is widely used in local search (W)PMS solvers. This section first introduces the main process of FPS, and then introduces how to use FPS to improve the state-of-the-art local search (W)PMS solvers.

#### 3.1 The Proposed FPS Strategy

As described in Section 2.2, the variable selection strategies (lines 5-6 in Algorithm 1) based on the single flipping mechanism are reasonable when local optima are not reached, since the current solution can always be improved. However, when the algorithms fall into a local optimum, they stop improving the current solution and allow the search to get a worse solution to escape from the local optimum. We argue that such a strategy may make the algorithms miss better solutions, and we propose FPS to handle this issue by using a two-level look-ahead technique. When a local optimum for the single flipping mechanism is reached, FPS first samples some first-level variables, and then tries to look-ahead from these variables to see whether flipping a pair of variables can improve the current solution. In this way, a local optimum for the single flipping mechanism might not be a local optimum for FPS. Moreover, FPS escapes from its local optima by selecting the best to flip among the best sampled first-level variable and the best sampled pair of variables, which can provide a better search direction than the widely-used random walk local optima escaping strategy.

The procedure of a general local search algorithm based on FPS is shown in Algorithm 2. Note that we use the single scoring function  $\text{score}(\cdot)$  to depict the procedure, which is easier to understand. For the algorithms that the scoring functions regarding hard and soft clauses are calculated independently, we just need to replace  $\text{score}(\cdot)$  with  $\text{hscore}(\cdot)$  and  $\text{sscore}(\cdot)$  accordingly.

When  $\text{GoodVars} \neq \emptyset$ , FPS also selects the variable to be flipped from *GoodVars* (lines 5-7) as Algorithm 1 does. When  $\text{GoodVars} = \emptyset$ , FPS first samples a set of first-level variables (lines 9-13), denoted as  $FV$ , and then look-ahead from each first-level variable (lines 16-27). To determine  $FV$ , the algorithm first samples  $sc\_num$  (10 by default) falsified hard clauses, if any; otherwise, samples  $sc\_num$  falsified soft clauses, and then randomly samples a first-level variable in each sampled clause. Such a sampling strategy for selecting  $FV$  is effective in FPS. On the one hand, look-ahead from all the variables in all the falsified clauses is time-consuming. Thus the sampling strategy can improve the efficiency. On the other hand, sampling first-level variables from multiple clauses can provide diverse search directions, *i.e.*, satisfying different clauses.

After sampling  $FV$ , the first-level variable with the highest  $\text{score}$  is recorded as  $v_1$ , whose  $\text{score}$  is recorded as  $s_1$  (lines 14-15). Then, FPS tries to perform a pseudo flipping for each first-level variable  $FV_i$  (line 18). Note that the pseudo flipping operator will not change the current solution and maintained information, just to look-ahead to determine  $\text{GoodVars}'$  (line 19), which records the second-level variables with a positive  $\text{score}$  after flipping  $FV_i$  (*i.e.*, the  $\text{score}$  of each second-level variable is the one computed after performing the pseudo flipping of  $FV_i$ ). Such a pseudo flipping operator can avoid the redundant flipping operator for restoring the current solution  $A$  after each look-ahead process.

If  $\text{GoodVars}' = \emptyset$ , which means flipping both  $FV_i$  and any second-level variable in  $\text{GoodVars}'$  will not be better than only flipping  $FV_i$ . In this case, look-ahead from  $FV_i$  can not gain further benefits, and the algorithm will

---

**Algorithm 2:** General Local Search based on FPS

---

**Input:** A (W)PMS instance  $\mathcal{F}$ , cut-off time  $cutoff$ , number of sampled clauses  $sc\_num$ , BMS parameter  $sv\_num$

**Output:** A feasible assignment  $A$  of  $\mathcal{F}$ , or no feasible assignment found

```
1  $A :=$  an initial assignment;  $A^* := A$ ;  
2 while running time  $<$   $cutoff$  do  
3   if  $A$  is feasible &  $cost(A) < cost(A^*)$  then  
4      $A^* := A$ ;  
5   if  $GoodVars := \{x | score(x) > 0\} \neq \emptyset$  then  
6      $v :=$  a variable selected from  $GoodVars$ ;  
7      $Vars := \{v\}$ ;  
8   else  
9      $SC :=$  the set of  $sc\_num$  randomly selected  
10    falsified clauses (with a bias to hard ones);  
11    The set of first-level variables  $FV := \emptyset$ ;  
12    for  $i := 1$  to  $sc\_num$  do  
13       $v :=$  a random variable in  $SC_i$ ;  
14      if  $v \notin FV$  then  $FV := FV \cup \{v\}$ ;  
15     $v_1 := \arg \max_{v \in FV} score(v)$ ;  
16     $s_1 := score(v_1)$ ;  $s_2 := -\infty$ ;  
17    for  $i := 1$  to  $|FV|$  do  
18       $score' := score(FV_i)$ ;  
19      Performing a pseudo flipping for  $FV_i$ ;  
20      if  $GoodVars' := \{x | score(x) > 0\} \neq \emptyset$   
21      then  
22         $SV_i :=$  a second-level variable in  
23         $GoodVars'$  picked by BMS with  
24        parameter  $sv\_num$ ;  
25        if  $score' + score(SV_i) > 0$  then  
26           $Vars := \{FV_i, SV_i\}$ ;  
27           $s_2 := score' + score(SV_i)$ ;  
28          break;  
29        if  $score' + score(SV_i) > s_2$  then  
30           $v_2^1 := FV_i$ ;  $v_2^2 := SV_i$ ;  
31           $s_2 := score' + score(SV_i)$ ;  
32      if  $s_1 > s_2$  then  $Vars := \{v_1\}$ ;  
33      else  $Vars := \{v_2^1, v_2^2\}$ ;  
34    $A := A$  with the variables in  $Vars$  flipped;  
35 if  $A^*$  is feasible then return  $A^*$ ;  
36 else return no feasible assignment found;
```

---

continue to look-ahead from the next first-level variable. If  $GoodVars' \neq \emptyset$ , flipping both  $FV_i$  and some second-level variable  $SV_i$  in  $GoodVars'$  might improve the current local optimal solution. FPS selects  $SV_i$  by a probabilistic sampling strategy called Best from Multiple Selections (BMS) (Cai 2015), which chooses  $sv\_num$  random variables from  $GoodVars'$  and returns the one with the highest  $score$  (line 20). Once a pair of variables ( $FV_i, SV_i$ ) that flipping both can improve the current local optimum is found, FPS uses an early-stop strategy (lines 21-24) to terminate

the traversing of  $FV$ . The best sampled pair of variables are recorded as  $(v_2^1, v_2^2)$ , and the total  $score$  of flipping them is recorded as  $s_2$  (lines 26-27). Finally, if FPS fails to improve the current solution, it selects the best to flip among  $v_1$  and  $(v_2^1, v_2^2)$  according to the benefits of flipping them, *i.e.*,  $s_1$  and  $s_2$  (lines 28-29).

### 3.2 Applying FPS to Various Local Searches

As shown in Algorithm 2, FPS actually provides a two-level look-ahead strategy for selecting the variables to be flipped when the local optima for the single flipping mechanism are reached. Therefore, a simple application of FPS to a local search (W)PMS algorithm is to replace its local optima escaping strategy (*e.g.*, lines 7-9 in Algorithm 1) with that in FPS (*i.e.*, lines 8-29 in Algorithm 2). In this work, we use this method to apply FPS to SATLike3.0 (Cai and Lei 2020), CCEHC (Luo et al. 2017), and Dist (Cai et al. 2014). Among them, SATLike3.0, the newest extension of the famous SATLike (Lei and Cai 2018) algorithm, is the best-performing one and also a typical algorithm based on the single flipping mechanism and the simple random walk local optima escaping strategy. Therefore, we choose SATLike3.0 as the core baseline algorithm. The resulting algorithm of applying FPS to SATLike3.0 is called MaxFPS (available at <https://github.com/JHL-HUST/FPS>). CCEHC and Dist are two representative algorithms for WPMS and PMS, respectively. We use CCEHC-FPS (resp. Dist-FPS) to represent the algorithm of applying FPS to CCEHC (resp. Dist).

We further apply FPS to improve the recently proposed local search (W)PMS algorithm, BandMaxSAT (Zheng et al. 2022). Since BandMaxSAT mainly uses its bandit model to select the search direction to escape from the local optima, we can not keep the core features of BandMaxSAT by simply replacing its local optima escaping strategy with that in FPS. Therefore, we propose another way to apply FPS to BandMaxSAT. That is, the first-level variables  $FV$  in FPS are sampled from the clause selected by the bandit model in BandMaxSAT. The resulting solver is called BandMaxSAT-FPS. The application of FPS in BandMaxSAT-FPS indicates that one can design appropriate methods to determine  $FV$  to use FPS flexibly.

## 4 Experiments

For experiments, we first analyze the influence of parameters  $sc\_num$  and  $sv\_num$  on the performance of FPS. Then we compare the algorithms improved by FPS with the baselines, *i.e.*, MaxFPS vs. SATLike3.0, CCEHC-FPS vs. CCEHC, Dist-FPS vs. Dist, and BandMaxSAT-FPS vs. BandMaxSAT, to evaluate the performance of FPS. We further replace the local search component in SATLike-c (Lei et al. 2021), *i.e.*, SATLike3.0, with MaxFPS, and compare the resulting solver MaxFPS-c with some of the state-of-the-art SAT-based (W)PMS solvers, including SATLike-c, TT-Open-WBO-Inc (Nadel 2019), and Loandra (Berg, Demirovic, and Stuckey 2019). Finally, we do ablation studies for clarity and in-depth analysis.

Note that since the typical SATLike3.0 algorithm based on the single flipping mechanism is our core baseline algo-

rithm, we use its improved algorithm MaxFPS to analyze the parameters and do ablation studies.

#### 4.1 Experimental Setup

For the (W)PMS experiments, we select all the instances from the incomplete track of the last four MaxSAT Evaluations (MSEs), *i.e.*, MSE2018 to MSE2021, for comparison and analysis. We use (W)PMS<sub>*y*</sub> to represent the (W)PMS benchmarks in MSE of year *y*. All the algorithms in the experiments were implemented in C++ and run on a server using an Intel® Xeon® E5-2650 v3 2.30 GHz CPU and 256 GB RAM, running Ubuntu 16.04 Linux operation system. Each (W)PMS instance is solved once (as the baselines and MSE do) by each algorithm with two time limits, 60 and 300 seconds, which are consistent with the settings in MSEs. The random seed for each algorithm is set to 1 as the baseline algorithms do. Due to the limited space, this section only presents the results within 300 seconds of time limit. See results within 60 seconds of time limit in Appendix.

#### 4.2 Parameter Study

The parameters in FPS include *sc\_num*, the number of sampled clauses, and *sv\_num*, the BMS parameter for sampling the second-level variables. We compare 12 different settings of  $sc\_num \in \{5, 10, 20, 50\}$  and  $sv\_num \in \{20, 50, 100\}$  (selected according to our experience) based on MaxFPS on all the (W)PMS instances from the incomplete track of MSE2017. We use the scoring function in MSEs to calculate a score for each algorithm per instance, which equals to zero if the output solution is infeasible, otherwise to the cost (see Section 2.1) of the best-known solution plus 1 divided by the cost of the output solution plus 1.

Figure 1 shows the comparison results of MaxFPS with different parameters. The results are expressed by the average score of all the tested instances in MSE2017. From the results, we can see that assigning moderate values to both *sc\_num* and *sv\_num* results in a good performance. Actually, the larger the value of *sc\_num* or *sv\_num*, the higher the quality of the local optima for MaxFPS, and the lower the algorithm efficiency. Therefore, moderate values of the parameters can well balance the search ability and algorithm efficiency. The results also demonstrate that the sampling strategies used in FPS are effective. We select the best among the 12 tested settings, *i.e.*,  $sc\_num = 10$  and  $sv\_num = 50$ , as the default parameters in all the algorithms improved by FPS in our experiments.

#### 4.3 Evaluation on FPS

This subsection first presents a comprehensive comparison between MaxFPS and SATLike3.0 to evaluate the performance of FPS, and then compares the other algorithms improved by FPS and the corresponding baselines to evaluate the generalization capability of FPS.

The comparison results of MaxFPS and SATLike3.0 are shown in Table 1. Column *#inst.* indicates the number of instances of each benchmark, column *#win.* indicates the number of instances in which the solver finds the best solution among all solvers in the table, and column *time* indicates the average time (in seconds) for obtaining the results

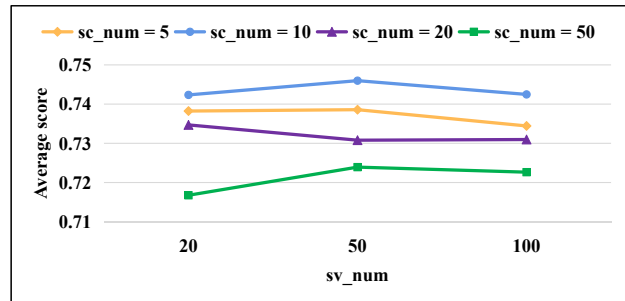


Figure 1: Comparison on MaxFPS with different settings of *sc\_num* and *sv\_num*.

Benchmark	#inst.	MaxFPS		SATLike3.0	
		#win.	time	#win.	time
PMS_2018	153	<b>111</b>	60.68	61	83.43
PMS_2019	299	<b>212</b>	49.05	142	56.64
PMS_2020	262	<b>189</b>	41.09	112	62.14
PMS_2021	155	<b>108</b>	47.12	64	51.00
WPMS_2018	172	<b>115</b>	78.90	64	84.37
WPMS_2019	297	<b>222</b>	94.71	100	80.38
WPMS_2020	253	<b>183</b>	92.49	86	72.82
WPMS_2021	151	<b>84</b>	104.92	64	98.31

Table 1: Comparison of MaxFPS and SATLike3.0.

of the *winning* instances. We could observe that the *winning* PMS (resp. WPMS) instances of MaxFPS are about 49-82% (resp. 31-122%) greater than those of SATLike3.0, indicating a significant improvement.

To obtain a more detailed comparison between MaxFPS and SATLike3.0 and evaluate the performance of FPS on different instance classes, we collect all the tested instances (duplicated ones are removed) and compare MaxFPS with SATLike3.0 on each instance class. Ties of these two algorithms with the same number of *winning* instances are broken by selecting the one with less running time (as the rules in MSEs). The results on the PMS and WPMS instance classes are shown in Tables 2 and 3, respectively. Note that we remove the instance classes that both MaxFPS and SATLike3.0 can not yield feasible solutions. The results show that MaxFPS outperforms SATLike3.0 on most classes of both PMS and WPMS instances. Specifically, for all the 34 (resp. 27) classes of PMS (resp. WPMS) instances, MaxFPS outperforms SATLike3.0 on 29 (resp. 20) classes, indicating the excellent robustness of FPS that can boost SATLike3.0 in solving various classes of (W)PMS instances.

The comparison results of CCEHC-FPS and CCEHC, Dist-FPS and Dist, BandMaxSAT-FPS (BandMS-FPS) and BandMaxSAT, are summarized in Tables 4, 5, and 6, respectively. The results show that these baselines can all be significantly improved by our FPS strategy, indicating its excellent generalization performance and robustness. Moreover, FPS can also significantly improve the SATLike3.0, CCEHC, Dist, and BandMaxSAT algorithms within 60 seconds of time limit (see details in Appendix).

Benchmark	#inst.	MaxFPS		SATLike3.0	
		#win.	time	#win.	time
aes	6	<b>5</b>	37.65	2	136.81
atcoss	14	<b>1</b>	272.39	0	0.00
decision-tree	23	<b>21</b>	17.76	6	115.36
extension-enforcement	19	14	96.19	<b>17</b>	100.30
gen-hyper-tw	37	26	<b>103.40</b>	26	110.05
hs-timetabling	1	<b>1</b>	1.59	0	0.00
large-graph-community	3	<b>3</b>	6.76	2	10.51
logic-synthesis	1	<b>1</b>	2.57	0	0.00
bcp	24	<b>22</b>	65.85	6	129.55
pseudoBoolean	11	<b>1</b>	296.65	0	0.00
maxclique & maxcut	58	<b>58</b>	12.87	54	1.42
MCS-GE	25	<b>25</b>	9.24	14	32.56
MaxSATQIC	35	<b>25</b>	40.29	19	55.75
min-fill	16	<b>11</b>	28.78	6	84.63
optic	17	<b>17</b>	39.89	0	0.00
phylogenetic-trees	11	<b>2</b>	131.40	0	0.00
railroad_reisch	9	<b>9</b>	39.14	5	6.57
railway-transport	4	<b>2</b>	34.92	1	298.83
ramsey	14	14	<b>0.04</b>	14	0.11
reversi	11	<b>2</b>	104.29	0	0.00
des	13	1	225.94	<b>2</b>	89.65
scheduling	5	2	84.98	<b>3</b>	154.90
scheduling_xiaojuan	8	<b>5</b>	127.53	4	85.77
set-covering	9	<b>9</b>	68.72	1	122.93
setcover-rail_zhendong	4	<b>4</b>	210.15	2	1.72
treewidth-computation	9	<b>9</b>	134.01	5	65.29
uaq	20	<b>20</b>	11.10	18	46.98
uaq_gazzarata	1	<b>1</b>	188.66	0	0.00
xai-mindset2	19	<b>16</b>	44.07	1	227.51
mbd	6	3	141.53	<b>5</b>	194.20
SeanSafarpour	13	<b>9</b>	141.25	8	142.53
fault-diagnosis	8	<b>7</b>	31.42	0	0.00
close_solutions	14	5	43.89	<b>9</b>	35.75
causal-discovery	3	3	<b>3.27</b>	3	5.40
Total	471	<b>354</b>	48.93	233	59.02

Table 2: Comparison of MaxFPS and SATLike3.0 on each PMS instance class. MCS-GE (resp. MaxSATQIC) is a short name of MaximumCommonSub-GraphExtraction (resp. MaxSATQueriesinInterpretableClassifiers).

#### 4.4 Comparison with SAT-based (W)PMS Solvers

We then replace the local search component in SATLike-c (*i.e.*, SATLike3.0) with MaxFPS, and compare the resulting solver MaxFPS-c with some of the state-of-the-art SAT-based (W)PMS solvers, including SATLike-c, TT-OpenWBO-Inc (TT-OWI), and Loandra. Note that the results of these SAT-based solvers are obtained by running their codes downloaded from MSE2021. We use the scoring function in MSEs (introduced in Section 4.2) to calculate a score for each solver per instance, and present the average score of each solver on each benchmark in Table 7. We also depict the distribution of scores per instance of these SAT-based solvers as in MSEs in Appendix. The results show that MaxFPS-c yields the highest average score on all the benchmarks. Moreover, FPS not only can improve SATLike-c, but also help it outperform the other SAT-based (W)PMS solvers. These results further indicate the excellent performance of FPS.

#### 4.5 Ablation Study

Finally, we do ablation studies to evaluate the effectiveness and rationality of components in FPS, including the local optima escaping method and the two-level look-ahead search

Benchmark	#inst.	MaxFPS		SATLike3.0	
		#win.	time	#win.	time
abstraction-refinement	10	<b>9</b>	175.95	2	170.69
af-synthesis	32	<b>28</b>	21.97	5	196.89
correlation-clustering	44	16	145.31	<b>31</b>	131.22
decision-tree	24	11	160.08	<b>17</b>	127.94
hs-timetabling	13	<b>8</b>	146.10	0	0.00
lisbon-wedding	21	<b>15</b>	132.27	0	0.00
maxcut	28	<b>28</b>	6.20	26	1.68
MaxSATQIC	32	<b>26</b>	81.67	9	48.93
metro	2	2	234.82	0	0.00
MWDSP	7	6	<b>90.34</b>	6	105.84
min-width	40	<b>39</b>	146.55	1	153.18
mpe	19	<b>19</b>	74.17	5	91.75
RBAC	54	23	140.27	<b>33</b>	134.35
railroad_reisch	6	<b>6</b>	103.38	1	17.14
railway-transport	4	1	46.68	<b>2</b>	107.02
ramsey	12	9	31.09	<b>11</b>	31.70
relational-inference	2	1	<b>73.91</b>	1	212.09
scSequencing_Mehrabadi	10	2	85.73	<b>8</b>	57.30
set-covering	13	<b>13</b>	68.63	1	42.34
staff-scheduling	11	<b>11</b>	154.20	0	0.00
spot5	5	<b>5</b>	82.66	0	0.00
causal-discovery	24	<b>23</b>	19.15	15	17.64
timetabling	19	<b>15</b>	187.30	0	0.00
max-realizability	13	<b>10</b>	18.43	8	65.04
BTBNSL-Rounded	26	<b>14</b>	22.85	12	64.80
tcp	13	<b>12</b>	128.59	2	241.91
cluster-expansion	20	7	0.04	<b>17</b>	81.70
Total	504	<b>359</b>	90.77	213	86.67

Table 3: Comparison of MaxFPS and SATLike3.0 on each WPMS instance class. MWDSP (resp. MaxSATQIC) is a short name of MinimumWeightDominatingSetProblem (resp. MaxSATQueriesinInterpretableClassifiers).

Benchmark	#inst.	CCEHC-FPS		CCEHC	
		#win.	time	#win.	time
WPMS_2018	172	<b>78</b>	101.40	58	85.49
WPMS_2019	297	<b>143</b>	114.06	94	98.32
WPMS_2020	253	<b>112</b>	129.33	89	129.43
WPMS_2021	151	46	157.44	<b>59</b>	136.29

Table 4: Comparison of CCEHC-FPS and CCEHC.

Benchmark	#inst.	Dist-FPS		Dist	
		#win.	time	#win.	time
PMS_2018	153	<b>87</b>	92.43	73	93.47
PMS_2019	299	<b>170</b>	62.73	152	77.47
PMS_2020	262	<b>145</b>	62.16	119	84.77
PMS_2021	155	<b>77</b>	87.17	74	71.68

Table 5: Comparison of Dist-FPS and Dist.

process, by comparing MaxFPS with its several variants.

To evaluate the effectiveness of the components, we first perform two groups of comparison. The first one compares MaxFPS with its variant, MaxFPS<sub>1</sub>, which replaces the local optima escaping strategy in MaxFPS (lines 28-29 in Algorithm 2) with the simple random walk strategy (lines 8-9 in Algorithm 1). The second one compares MaxFPS<sub>1</sub> with SATLike3.0. The results of these two groups of comparison are shown in Tables 8 and 9, respectively. We can observe that MaxFPS significantly outperforms MaxFPS<sub>1</sub>, indicating that the local optima escaping strategy in FPS is signif-

Benchmark	#inst.	BandMS-FPS		BandMaxSAT	
		#win.	time	#win.	time
PMS_2018	153	<b>105</b>	70.91	79	102.39
PMS_2019	299	<b>203</b>	68.81	158	67.44
PMS_2020	262	<b>172</b>	65.39	135	80.15
PMS_2021	155	<b>91</b>	74.66	89	67.78
WPMS_2018	172	<b>102</b>	105.83	73	104.27
WPMS_2019	297	<b>179</b>	98.49	149	89.74
WPMS_2020	253	<b>153</b>	111.28	121	122.00
WPMS_2021	151	<b>83</b>	132.67	62	133.11

Table 6: Comparison of BandMS-FPS and BandMaxSAT.

Benchmark	MaxFPS-c	SATLike-c	TT-OWI	Loandra
PMS_2018	<b>0.8682</b>	0.8399	0.8371	0.7912
PMS_2019	<b>0.8796</b>	0.8734	0.8698	0.7919
PMS_2020	<b>0.8696</b>	0.8457	0.8534	0.8156
PMS_2021	<b>0.8802</b>	0.8508	0.8407	0.8127
WPMS_2018	<b>0.9084</b>	0.8931	0.8994	0.8752
WPMS_2019	<b>0.9091</b>	0.8810	0.9010	0.8321
WPMS_2020	<b>0.8868</b>	0.8609	0.8650	0.8312
WPMS_2021	<b>0.8101</b>	0.7843	0.7758	0.7945

Table 7: Comparison of MaxFPS-c and complete solvers.

icantly better than the random walk strategy that is widely used in recent local search (W)PMS algorithms. This is because our method can provide more and better directions to escape from local optima. Moreover, MaxFPS<sub>1</sub> significantly outperforms SATLike3.0, indicating that only using the two-level look-ahead method can also significantly improve the local search algorithms, by improving the local optimal solutions of the single flipping mechanism.

Then, we compare MaxFPS with its other two variants, MaxFPS<sub>2</sub> and MaxFPS<sub>3</sub>, to analyze the rationality of the two-level search process in MaxFPS. MaxFPS<sub>2</sub> extends the two-level search approach in MaxFPS to three levels. In MaxFPS<sub>2</sub>, the third-level variables are selected by the same approach for selecting the second-level variables as in MaxFPS, and the second-level variables in MaxFPS<sub>2</sub> are sampled from the neighbors (variables that appeared in the same clause) of the corresponding first-level variable. MaxFPS<sub>3</sub> is a variant of MaxFPS that also look-ahead when the current solution is not a local optimum for the single flipping mechanism, *i.e.*,  $GoodVars \neq \emptyset$ . The results of MaxFPS and these two variants are summarised in Table 10.

From the results in Table 10, we can see that MaxFPS significantly outperforms MaxFPS<sub>2</sub>. This is because the three-level look-ahead process is too time-consuming and inefficient. Thus the proposed two-level look-ahead technique in FPS is reasonable. We can also observe that MaxFPS outperforms MaxFPS<sub>3</sub>, demonstrating that look-ahead only when  $GoodVars = \emptyset$  is reasonable and effective.

## 5 Conclusion

In this work, we propose an effective and general strategy to replace the mechanism of flipping a single variable per iteration that is widely used in local search MaxSAT solvers.

Benchmark	#inst.	MaxFPS		MaxFPS <sub>1</sub>	
		#win.	time	#win.	time
PMS_2018	153	<b>104</b>	71.21	75	82.91
PMS_2019	299	<b>202</b>	55.93	172	55.63
PMS_2020	262	<b>163</b>	48.37	140	67.44
PMS_2021	155	<b>98</b>	53.10	89	46.67
WPMS_2018	172	<b>119</b>	69.45	69	49.45
WPMS_2019	297	<b>212</b>	92.88	123	77.06
WPMS_2020	253	<b>177</b>	90.82	104	83.44
WPMS_2021	151	<b>89</b>	107.00	61	83.44

Table 8: Comparison of MaxFPS and MaxFPS<sub>1</sub>.

Benchmark	#inst.	MaxFPS <sub>1</sub>		SATLike3.0	
		#win.	time	#win.	time
PMS_2018	153	<b>106</b>	72.42	63	83.74
PMS_2019	299	<b>206</b>	51.51	142	53.88
PMS_2020	262	<b>161</b>	53.66	112	75.61
PMS_2021	155	<b>105</b>	46.68	68	62.79
WPMS_2018	172	90	74.04	<b>92</b>	90.23
WPMS_2019	297	<b>188</b>	88.56	131	96.44
WPMS_2020	253	<b>156</b>	90.30	116	98.30
WPMS_2021	151	71	86.66	<b>75</b>	109.47

Table 9: Comparison of MaxFPS<sub>1</sub> and SATLike3.0.

Benchmark	#inst.	MaxFPS		MaxFPS <sub>2</sub>		MaxFPS <sub>3</sub>	
		#win.	time	#win.	time	#win.	time
PMS_2018	153	<b>96</b>	78.45	44	20.69	80	59.52
PMS_2019	299	<b>194</b>	62.70	103	13.05	166	46.30
PMS_2020	262	<b>148</b>	50.68	86	16.96	136	63.19
PMS_2021	155	<b>89</b>	59.75	58	13.01	85	39.88
WPMS_2018	172	<b>85</b>	66.54	57	13.23	70	74.63
WPMS_2019	297	<b>181</b>	91.87	92	13.28	144	78.45
WPMS_2020	253	<b>164</b>	94.40	61	12.52	112	75.66
WPMS_2021	151	<b>83</b>	118.80	32	21.86	51	103.99

Table 10: Comparison of MaxFPS, MaxFPS<sub>2</sub>, MaxFPS<sub>3</sub>.

The proposed FPS strategy combines the look-ahead technique and probabilistic sampling method. As a result, FPS can improve the local optimum of the single flipping mechanism and provide more and better search directions to escape from local optima.

Look-ahead is not a new technique, but how to look-ahead is the magic recipe. This work proposes an effective way to apply this technique to boost local search MaxSAT solvers. We also demonstrate that there is great potential for the look-ahead technique to be used for MaxSAT. Extensive experiments demonstrate that our proposed FPS strategy significantly improves the state-of-the-art (W)PMS solvers, and FPS has an excellent generalization performance to various local search MaxSAT solvers, including SATLike3.0, CCEHC, Dist, and BandMaxSAT as well as one of the state-of-the-art SAT-based solvers, SATLike-c.

Moreover, we have also tried to apply FPS to local search SAT solvers such as CCAnr (Cai, Luo, and Su 2015) and obtained promising performance. In the future, we will further explore the potential of FPS in MaxSAT and SAT solving.

## Acknowledgments

This work is supported by National Natural Science Foundation (U22B2017) and MSRA Collaborative Research 2022 (100338928).

## References

- Berg, J.; Demirovic, E.; and Stuckey, P. J. 2019. Core-Boosted Linear Search for Incomplete MaxSAT. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 11494, 39–56.
- Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 2703–2710.
- Cai, S. 2015. Balance between Complexity and Quality: Local Search for Minimum Vertex Cover in Massive Graphs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 747–753.
- Cai, S.; and Lei, Z. 2020. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287: 103354.
- Cai, S.; Luo, C.; and Su, K. 2015. CCAnr: A Configuration Checking Based Local Search Solver for Non-random Satisfiability. In *Proceedings of the Eighteenth International Conference of Theory and Applications of Satisfiability Testing*, volume 9340, 1–8.
- Cai, S.; Luo, C.; Thornton, J.; and Su, K. 2014. Tailoring Local Search for Partial MaxSAT. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2623–2629.
- Cai, S.; Luo, C.; and Zhang, H. 2017. From Decimation to Local Search and Back: A New Approach to MaxSAT. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 571–577.
- Cai, S.; and Su, K. 2013. Local search for Boolean Satisfiability with configuration checking and subscore. *Artificial Intelligence*, 204: 75–98.
- Cai, S.; Su, K.; and Luo, C. 2013. Improving WalkSAT for Random  $k$ -Satisfiability Problem with  $k > 3$ . In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 145–151.
- Cai, S.; Su, K.; and Sattar, A. 2011. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9-10): 1672–1696.
- Cha, B.; Iwama, K.; Kambayashi, Y.; and Miyazaki, S. 1997. Local Search Algorithms for Partial MAXSAT. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 263–268.
- Ciampiconi, L.; Ghosh, B.; Scarlett, J.; and Meel, K. S. 2020. A MaxSAT-Based Framework for Group Testing. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 10144–10152.
- Demirovic, E.; and Musliu, N. 2017. MaxSAT-based large neighborhood search for high school timetabling. *Computers & Operations Research*, 78: 172–180.
- Khadilkar, H. 2022. Solving the capacitated vehicle routing problem with timing windows using rollouts and MAXSAT. arXiv:2206.06618.
- Lei, Z.; and Cai, S. 2018. Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 1346–1352.
- Lei, Z.; Cai, S.; Geng, F.; Wang, D.; Peng, Y.; Wan, D.; Deng, Y.; and Lu, P. 2021. SATLike-c: Solver Description. *MaxSAT Evaluation 2021*, 19–20.
- Li, C. M.; and Huang, W. 2005. Diversification and Determinism in Local Search for Satisfiability. In *Proceedings of the Eighth International Conference of Theory and Applications of Satisfiability Testing*, volume 3569, 158–172.
- Li, C. M.; Wei, W.; and Zhang, H. 2007. Combining Adaptive Noise and Look-Ahead in Local Search for SAT. In *Proceedings of the Tenth International Conference of Theory and Applications of Satisfiability Testing*, volume 4501, 121–133.
- Luo, C.; Cai, S.; Su, K.; and Huang, W. 2017. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243: 26–44.
- Mali, A. D.; and Lipen, Y. 2003. MFSAT: A SAT Solver Using Multi-Flip Local Search. In *Proceedings of the Fifteenth IEEE International Conference on Tools with Artificial Intelligence*, 84–93.
- Morris, P. 1993. The Breakout Method for Escaping from Local Minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 40–45.
- Nadel, A. 2019. Anytime weighted MaxSAT with improved polarity selection and bit-vector optimization. In *2019 Formal Methods in Computer Aided Design*, 193–202.
- Reisch, J.; Großmann, P.; and Kliewer, N. 2020. Stable Resolving - A Randomized Local Search Heuristic for MaxSAT. In *Proceedings of the Forty-Third German Conference on Artificial Intelligence*, volume 12325, 163–175.
- Selman, B.; Kautz, H. A.; and Cohen, B. 1993. Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop*, volume 26, 521–531.
- Wei, W.; Li, C. M.; and Zhang, H. 2008. A Switching Criterion for Intensification and Diversification in Local Search for SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4): 219–237.
- Zheng, J.; He, K.; Zhou, J.; Jin, Y.; Li, C. M.; and Manyà, F. 2022. BandMaxSAT: A Local Search MaxSAT Solver with Multi-armed Bandit. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, 1901–1907.



## Appendix

In the Appendix, we mainly present some supplementary experiments. We first present the comprehensive comparison results between the algorithms improved by FPS and the local search (W)PMS baselines within two time limits, 60 and 300 seconds, as in MSEs. Then, we present the detailed comparison of MaxFPS-c and the SAT-based solvers. Finally, we present some additional ablation studies.

Benchmarks used in Appendix are the same as those described in Section 4.1, *i.e.*, all the (W)PMS instances from the incomplete track of the last four MSEs.

### A Comprehensive Evaluation on FPS

The comparison results, within 60 or 300 seconds of time limit, of MaxFPS and SATLike3.0 (Cai and Lei 2020), CCEHC-FPS and CCEHC (Luo et al. 2017), Dist-FPS and Dist (Cai et al. 2014), BandMaxSAT-FPS (BandMS-FPS) and BandMaxSAT (Zheng et al. 2022), are summarized in

Benchmark	#inst.	MaxFPS		SATLike3.0	
		#win.	time	#win.	time
60 seconds of time limit					
PMS_2018	153	<b>104</b>	13.69	49	15.50
PMS_2019	299	<b>199</b>	11.52	132	11.87
PMS_2020	262	<b>180</b>	12.03	100	10.88
PMS_2021	155	<b>103</b>	9.56	57	8.63
WPMS_2018	172	<b>118</b>	16.90	62	11.97
WPMS_2019	297	<b>209</b>	20.15	94	17.19
WPMS_2020	253	<b>167</b>	19.57	84	20.85
WPMS_2021	151	<b>73</b>	25.75	58	27.66
300 seconds of time limit					
PMS_2018	153	<b>111</b>	60.68	61	83.43
PMS_2019	299	<b>212</b>	49.05	142	56.64
PMS_2020	262	<b>189</b>	41.09	112	62.14
PMS_2021	155	<b>108</b>	47.12	64	51.00
WPMS_2018	172	<b>115</b>	78.90	64	84.37
WPMS_2019	297	<b>222</b>	94.71	100	80.38
WPMS_2020	253	<b>183</b>	92.49	86	72.82
WPMS_2021	151	<b>84</b>	104.92	64	98.31

Table 11: Comparison of MaxFPS and SATLike3.0.

Benchmark	#inst.	CCEHC-FPS		CCEHC	
		#win.	time	#win.	time
60 seconds of time limit					
WPMS_2018	172	<b>77</b>	25.99	57	21.11
WPMS_2019	297	<b>138</b>	27.34	82	21.94
WPMS_2020	253	<b>96</b>	25.77	79	26.82
WPMS_2021	151	40	32.42	<b>48</b>	34.30
300 seconds of time limit					
WPMS_2018	172	<b>78</b>	101.40	58	85.49
WPMS_2019	297	<b>143</b>	114.06	94	98.32
WPMS_2020	253	<b>112</b>	129.33	89	129.43
WPMS_2021	151	46	157.44	<b>59</b>	136.29

Table 12: Comparison of CCEHC-FPS and CCEHC.

Tables 11, 12, 13, and 14, respectively. The results show that, within either 60 or 300 seconds of time limit, FPS can significantly improve these state-of-the-art local search (W)PMS algorithms, indicating its excellent performance and robustness.

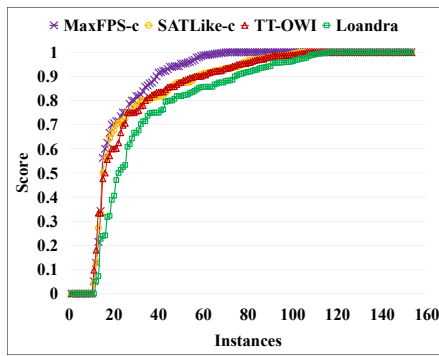
We also compare MaxFPS and SATLike3.0 within 60 seconds of time limit on each instance class. The results on the PMS and WPMS instance classes are shown in Tables 15 and 16, respectively. Note that duplicated instances and the instance classes that both MaxFPS and SATLike3.0 can not yield feasible solutions within 60 seconds are removed. The results show that, within 60 seconds of time limit, MaxFPS also outperforms SATLike3.0 on most classes of both PMS and WPMS instances. Specifically, for all the 32 (resp. 27) classes of PMS (resp. WPMS) instances, MaxFPS outperforms SATLike3.0 on 25 (resp. 20) classes, indicating again the excellent robustness of FPS that can boost SATLike3.0 in solving various classes of (W)PMS instances.

Benchmark	#inst.	Dist-FPS		Dist	
		#win.	time	#win.	time
60 seconds of time limit					
PMS_2018	153	<b>87</b>	20.70	67	19.25
PMS_2019	299	<b>168</b>	14.16	146	17.23
PMS_2020	262	<b>144</b>	14.50	110	17.52
PMS_2021	155	<b>76</b>	18.19	69	11.44
300 seconds of time limit					
PMS_2018	153	<b>87</b>	92.43	73	93.47
PMS_2019	299	<b>170</b>	62.73	152	77.47
PMS_2020	262	<b>145</b>	62.16	119	84.77
PMS_2021	155	<b>77</b>	87.17	74	71.68

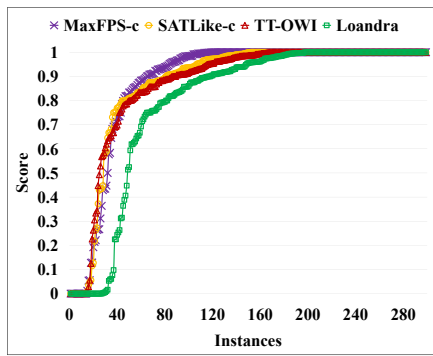
Table 13: Comparison of Dist-FPS and Dist.

Benchmark	#inst.	BandMS-FPS		BandMaxSAT	
		#win.	time	#win.	time
60 seconds of time limit					
PMS_2018	153	<b>91</b>	15.70	76	19.62
PMS_2019	299	<b>182</b>	14.36	157	15.43
PMS_2020	262	<b>160</b>	13.47	134	17.76
PMS_2021	155	79	12.69	<b>91</b>	17.61
WPMS_2018	172	<b>102</b>	20.43	82	20.44
WPMS_2019	297	<b>165</b>	20.15	151	23.16
WPMS_2020	253	<b>137</b>	21.96	122	23.66
WPMS_2021	151	<b>72</b>	29.78	63	30.57
300 seconds of time limit					
PMS_2018	153	<b>105</b>	70.91	79	102.39
PMS_2019	299	<b>203</b>	68.81	158	67.44
PMS_2020	262	<b>172</b>	65.39	135	80.15
PMS_2021	155	<b>91</b>	74.66	89	67.78
WPMS_2018	172	<b>102</b>	105.83	73	104.27
WPMS_2019	297	<b>179</b>	98.49	149	89.74
WPMS_2020	253	<b>153</b>	111.28	121	122.00
WPMS_2021	151	<b>83</b>	132.67	62	133.11

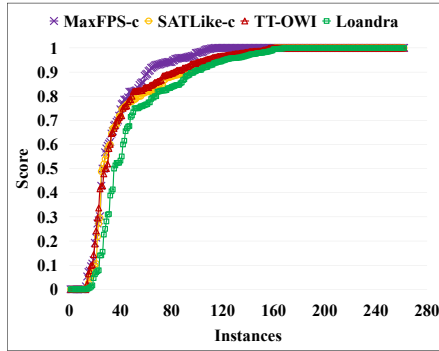
Table 14: Comparison of BandMS-FPS and BandMaxSAT.



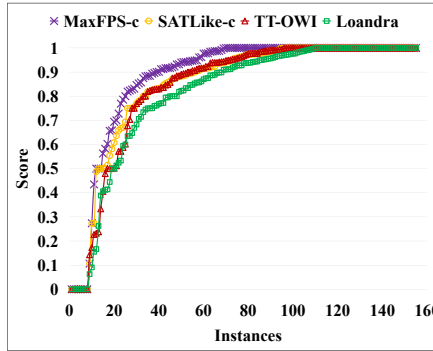
(a) Comparison on PMS\_2018



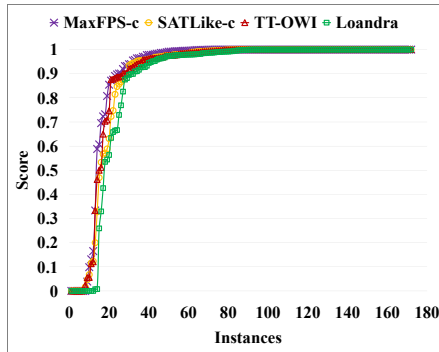
(b) Comparison on PMS\_2019



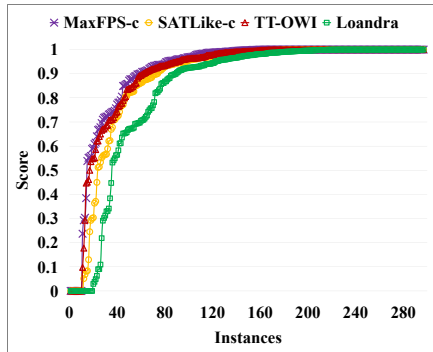
(c) Comparison on PMS\_2020



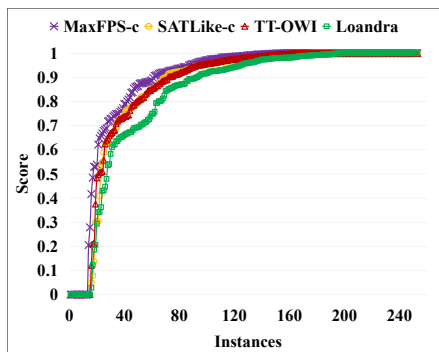
(d) Comparison on PMS\_2021



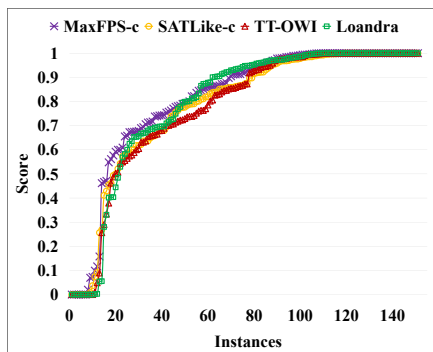
(e) Comparison on WPMS\_2018



(f) Comparison on WPMS\_2019



(g) Comparison on WPMS\_2020



(h) Comparison on WPMS\_2021

Figure 2: Distribution of scores per instance of MaxFPS-c, SATLike-c, TT-OWI, and Loandra on (W)PMS benchmarks.

Benchmark	#inst.	MaxFPS		SATLike3.0	
		#win.	time	#win.	time
aes	6	<b>4</b>	8.40	1	18.91
decision-tree	23	<b>20</b>	7.54	3	10.27
extension-enforcement	19	14	<b>14.86</b>	14	19.44
gen-hyper-tw	37	29	19.28	<b>20</b>	15.90
hs-timetabling	1	<b>1</b>	1.59	0	0.00
large-graph-community	3	<b>3</b>	6.76	2	10.51
logic-synthesis	1	<b>1</b>	2.57	0	0.00
bcp	24	<b>19</b>	21.06	5	35.71
pseudoBoolean	11	<b>1</b>	2.87	0	0.00
maxclique & maxcut	58	55	3.44	55	<b>1.40</b>
MCS-GE	25	<b>25</b>	7.97	11	3.53
MaxSATQIC	35	<b>27</b>	12.55	18	8.98
min-fill	16	<b>10</b>	15.82	6	32.44
optic	17	<b>15</b>	11.18	2	12.73
phylogenetic-trees	11	<b>1</b>	36.80	0	0.00
railroad_reisch	9	<b>9</b>	9.73	6	5.67
railway-transport	4	<b>2</b>	34.92	1	37.35
ramsey	14	14	<b>0.04</b>	14	0.11
des	13	1	59.40	<b>1</b>	<b>36.59</b>
scheduling	5	<b>3</b>	38.29	2	29.83
scheduling_xiaojuan	8	<b>6</b>	22.53	4	34.80
set-covering	9	<b>9</b>	19.75	0	0.00
setcover-rail_zhendong	4	2	<b>1.53</b>	2	1.72
treewidth-computation	9	5	7.94	<b>7</b>	20.15
uaq	20	<b>20</b>	8.04	11	12.86
uaq_gazzarata	1	<b>1</b>	24.91	0	0.00
xai-mindset2	19	<b>14</b>	6.63	1	30.61
mbd	6	2	23.39	<b>4</b>	15.88
SeanSafarpour	13	8	15.28	<b>10</b>	17.51
fault-diagnosis	8	<b>7</b>	31.42	0	0.00
close_solutions	14	5	5.61	<b>9</b>	18.08
causal-discovery	3	<b>3</b>	<b>3.27</b>	3	5.40
Total	446	<b>336</b>	11.49	212	11.22

Table 15: Comparison of MaxFPS and SATLike3.0 within 60 seconds of time limit on each PMS instance class. MCS-GE (resp. MaxSATQIC) is a short name of MaximumCommonSub-GraphExtraction (resp. MaxSAT-QueriesinInterpretableClassifiers).

## B Comparison with SAT-based Solvers

We then present detailed comparison results of MaxFPS-c and the state-of-the-art SAT-based (W)PMS solvers, SATLike-c (Lei et al. 2021), TT-Open-WBO-Inc (TT-OWI) (Nadel 2019), and Loandra (Berg, Demirovic, and Stuckey 2019), within 300 seconds of time limit, in Figure 2. The results are expressed by the distributions of scores per instance of these SAT-based solvers as in MSEs<sup>1</sup>. To draw the results for each solver per benchmark, we first sort the scores obtained by the solver in solving the instances in the benchmark in ascending order, and then use a point to record each score and connect the points by a smooth curve.

The results in Figure 2 show that the curves of MaxFPS-c are usually above the curves of the other three SAT-based solvers, indicating that in solving most of the instances, MaxFPS-c can yield better results.

## C Additional Ablation Study

Finally, we compare MaxFPS with its variant MaxFPS<sub>4</sub> to analyze the effectiveness of the early-stop strategy used in FPS (see Section 3.1). MaxFPS<sub>4</sub> is a variant of MaxFPS

<sup>1</sup><https://maxsat-evaluations.github.io/2019/results/incomplete/weighted-300s/summary.html>

Benchmark	#inst.	MaxFPS		SATLike3.0	
		#win.	time	#win.	time
abstraction-refinement	10	<b>5</b>	44.81	2	58.29
af-synthesis	32	<b>32</b>	3.70	1	0.21
correlation-clustering	44	21	20.18	<b>33</b>	22.05
decision-tree	24	4	30.97	<b>20</b>	38.12
hs-timetabling	13	<b>7</b>	44.19	0	0.00
lisbon-wedding	21	<b>15</b>	34.76	0	0.00
maxcut	28	<b>27</b>	0.86	26	1.68
MaxSATQIC	32	<b>23</b>	18.88	10	18.06
metro	2	1	55.85	1	<b>48.51</b>
MWDSP	7	2	<b>31.05</b>	2	38.79
min-width	40	<b>39</b>	36.33	1	47.49
mpe	19	<b>18</b>	1.86	3	27.52
RBAC	54	24	22.74	<b>34</b>	27.64
railroad_reisch	6	<b>6</b>	39.91	1	17.14
railway-transport	4	<b>2</b>	52.84	1	59.01
ramsey	12	9	5.29	<b>11</b>	7.79
relational-inference	2	<b>1</b>	58.97	0	0.00
scSequencing_Mehrabadi	10	2	33.86	<b>6</b>	14.39
set-covering	13	<b>13</b>	12.44	1	42.34
staff-scheduling	11	<b>10</b>	29.94	1	39.73
spot5	5	<b>5</b>	43.68	0	0.00
causal-discovery	24	<b>23</b>	4.44	15	4.56
timetabling	19	<b>13</b>	40.21	1	43.44
max-realizability	13	<b>10</b>	10.26	4	11.64
BTBNSL-Rounded	26	<b>14</b>	0.40	12	17.06
tcp	13	<b>10</b>	22.53	4	15.52
cluster-expansion	20	12	0.04	<b>14</b>	0.08
Total	504	<b>348</b>	18.53	204	18.55

Table 16: Comparison of MaxFPS and SATLike3.0 within 60 seconds of time limit on each WPMS instance class. MWDSP (resp. MaxSATQIC) is a short name of MinimumWeightDominatingSetProblem (resp. MaxSAT-QueriesinInterpretableClassifiers).

Benchmark	#inst.	MaxFPS		MaxFPS <sub>4</sub>	
		#win.	time	#win.	time
PMS_2018	153	<b>106</b>	71.87	85	57.14
PMS_2019	299	<b>198</b>	59.43	186	47.53
PMS_2020	262	<b>177</b>	46.64	161	41.79
PMS_2021	155	<b>96</b>	45.84	90	46.69
WPMS_2018	172	<b>109</b>	60.61	92	59.30
WPMS_2019	297	<b>188</b>	86.85	164	89.23
WPMS_2020	253	<b>160</b>	85.17	133	91.40
WPMS_2021	151	<b>78</b>	105.94	76	97.43

Table 17: Comparison of MaxFPS and MaxFPS<sub>4</sub>.

without early-stop strategy, *i.e.*, MaxFPS<sub>4</sub> does not stop traversing the first-level variables when it finds a pair of variables that flipping both can improve the current solution. The comparison results of MaxFPS and MaxFPS<sub>4</sub> within 300 seconds of time limit are shown in Table 17. From the results, we could observe that MaxFPS outperforms MaxFPS<sub>4</sub>, indicating that the early-stop strategy in FPS is reasonable and effective, because it can improve the efficiency.