Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT

Zhendong Lei and **Shaowei Cai***
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
School of Computer and Control Engineering, University of Chinese Academy of Sciences
{leizd,caisw}@ios.ac.cn

Abstract

Partial MaxSAT (PMS) generalizes SAT and MaxSAT by introducing hard clauses and soft clauses. PMS and Weighted PMS (WPMS) have many important real world applications. Local search is one popular method for solving (W)PMS. Recent studies on specialized local search for (W)PMS have led to significant improvements. But such specialized algorithms are complicated with the concepts tailored for hard and soft clauses. In this work, we propose a dynamic local search algorithm, which exploits the structure of (W)PMS by a carefully designed clause weighting scheme. Our solver SATLike adopts a local search framework for SAT and does not need any specialized concept for (W)PMS. Experiments on PMS and WPMS benchmarks from the MaxSAT Evaluations (MSE) 2016 and 2017 show that SATLike significantly outperforms state of the art local search solvers. Also, SATLike significantly narrows the gap between the performance of local search solvers and complete solvers on industrial benchmarks, and performs better than state of the art complete solvers on the MSE2017 benchmarks.

1 Introduction

Maximum Satisfiability (MaxSAT) is an optimization version of the famous Satisfiability (SAT) problem. Given a propositional formula expressed in the Conjunctive Normal Form (CNF), MaxSAT concerns about satisfying as many clauses as possible. Partial MaxSAT (PMS) is a generalization of MaxSAT, whose clauses are divided into hard and soft clauses and the goal is to find an assignment that satisfies all the hard clauses and maximize the number of satisfied soft clauses. One can also associate weights to the soft clauses, and the resulting problem is Weighted Partial MaxSAT (WPMS).

PMS and WPMS have many important real world applications, and there is intense interest in developing efficient solvers for these problems. Solvers for (W)PMS can be divided into complete and incomplete solvers. Complete solvers have gain great progress in the last decade. Particularly, a complete method based on iteratively calling a SAT solver, usually referred to as SAT-based method, have attracted much interest. Since its first introduction [Fu and Malik, 2006], many works have been done in improving this method [Davies and Bacchus, 2011; Ansótegui *et al.*, 2013; Morgado *et al.*, 2013; Martins *et al.*, 2014a; Narodytska and Bacchus, 2014; Ansótegui and Gabàs, 2017]. SAT-based solvers are especially well known for their good performance on industrial instances. Researchers also modify the complete solvers to make them incomplete (i.e., by returning better assignments when found). The main incomplete method is local search. Local search algorithms maintain a complete assignment and iteratively modify the assignment (i.e., flipping the value of a variable). They keep track of the best solution found throughout the search and return it when reaching the termination condition such as time limit. Local search for (W)PMS has stagnated for years until the recent improvements brought by new techniques [Cai *et al.*, 2014; 2016; Luo *et al.*, 2017; Cai *et al.*, 2017]. Another incomplete method is based on the relation of MaxSAT and the problem of Minimal Correction Subset (MCS) Extraction. It has been shown that restricted enumeration of MCSes is effective at approximating MaxSAT [Marques-Silva *et al.*, 2013; Grégoire *et al.*, 2014; Bacchus *et al.*, 2014; Mencía *et al.*, 2015]. Such MCS-based solvers also rely on making calls to complete SAT solvers, but they are usually less effective than the SAT-based solvers on structured instances.

Local search is essentially an anytime method and is particularly suitable for real domains where we are often interested in getting a good-quality solution within reasonable time. In many problem domains of combinatorial optimization, local search is widely acknowledged as one of the most effective methods for solving large instances, including those from real world applications. Comparatively, the way of local search for (W)PMS is more difficult and is more interesting, mainly due to the problem feature. Local search has witnessed its success on solving random and some crafted benchmarks, but it has been criticized for its poor performance on industrial instances for a long time. Recently, Cai et. al. firstly developed a local search solver that achieves state of the art performance when compared with complete MaxSAT solvers on unweighted MaxSAT industrial instances [Cai *et al.*, 2017]. Improvements have also been made on improving local search for

*Corresponding author

(W)PMS industrial instances, but the gap between the performance of local search and complete solvers is still considerably large on such benchmarks.

Related works in local search for (W)PMS can be categorized into two lines. Early studies belong to the first line, and these algorithms encode (W)PMS as weighted MaxSAT by introducing a $hard\ clause\ multiplier$, such that the weighted cost of a hard clause is the product of the multiplier and the actual weight currently added to the clause. The focus of this line is to set an appropriate value to the multiplier, which can be done statically [Cha *et al.*, 1997; Luo *et al.*, 2015] or dynamically [Thornton and Sattar, 1998; Thornton *et al.*, 2002]. Such solvers do not sufficiently exploit the structure of (W)PMS and has poor performance. Recent studies on specialized local search for (W)PMS emerge as the second paradigm. These local search algorithms introduce the concepts of hard score and soft score [Cai *et al.*, 2014], and employ different heuristics for hard and soft clauses respectively. The resulting solvers, including Dist [Cai *et al.*, 2014] and CCEHC [Luo *et al.*, 2017], as well as their improved versions [Cai *et al.*, 2016; 2017], made significant improvements over previous local search solvers. However, they are more complicated with the concepts and heuristics tailored for hard and soft clauses separately.

In this work, we propose a novel local search algorithm for (W)PMS, which adopts a dynamic local search framework for SAT and exploits the distinction of hard and soft clauses by a carefully designed clause weighting scheme. Briefly speaking, the weighting scheme updates weights for both hard and soft clauses, while it puts more increments to hard clauses each time and also sets a limit on the maximum weight that each soft clause can get. It is interesting to note that, previous local search algorithms for (W)PMS did not benefit that much from the techniques for SAT [Cai *et al.*, 2016]. One may consider that due to some reasons, e.g., (W)PMS contains hard and soft clauses, local search for SAT is not suitable for solving (W)PMS. In this work, we show that how local search techniques for SAT can be used for effectively solving (W)PMS. Our algorithm is thus called SATLike. According to experiments on PMS and WPMS benchmarks, SATLike significantly improves the state of the art of local search for (W)PMS on all benchmarks. Further, SATLike significantly narrows the gap between local search and complete solvers on industrial instances. Note that, these achievements are obtained by using simple concepts already used in local search for SAT.

The reminder of this paper is organized as follows. The next section introduces preliminary knowledge. We introduce our main ideas in Section 3. Then, we present our SATLike algorithm for (W)PMS in Section 4. Experimental results of our solver are presented in Section 5. Finally, we conclude the paper.

2 Preliminaries

Given a set of Boolean variables$\{x_1, x_2, x_3, \ldots, x_n\}$, a literal is either a variables x_i or its negation $\neg x_i$. A clause is a disjunction of literals (i.e., $C_i = l_{i1} \vee l_{i2} \vee \cdots \vee l_{ij}$). A conjunctive normal form (CNF) formula F is a conjunction of clauses (i.e., $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$), and can be expressed as a set of clauses.

For a formula F, the set of variables appeared in F is denoted by $Var(F)$. An assignment α is a mapping $Var(F) \rightarrow \{0, 1\}$. A complete assignment is a mapping that assigns to each variable either 0 or 1. A clause is satisfied if it has at least one true literal, and falsified if all the literals in the clause are false literals.

Given a CNF formula, the Partial MaxSAT (PMS) problem, in which some clauses are declared to be hard and the rest are declared to be soft, is the problem of finding an assignment such that all hard clauses are satisfied and the number of satisfied (resp. falsified) clauses is maximized (resp. minimized). In Weighted PMS (WPMS), each soft clause is associated with a positive integer as its weight, and the goal is to satisfy all hard clauses and maximize the total weight of satisfied soft clauses.

For a (W)PMS instance F, we say an assignment α is feasible iff it satisfies all hard clauses in F. The *cost* of a feasible assignment α is defined to be the number (resp. the sum of original weights) of falsified soft clauses under α for PMS (resp. WPMS). For convenience, the cost of any infeasible assignment is defined to be $+\infty$.

Local search algorithms for SAT and MaxSAT problems, including (W)PMS, maintain an assignment and modify it iteratively. Specifically, starting with a complete assignment, a local search algorithm chooses a variable and flips it in each subsequent step, trying to find a feasible assignment with a lower cost. Local search algorithms with clause weighting techniques modify the weights of clauses during the search, and are also called Dynamic Local Search (DLS). To distinguish the original weight and the weight maintained by clause weighting techniques, we use $org\_w(c)$ to denote the former, and $w(c)$ the latter. Local search algorithms use scoring functions and heuristics to guide the search. The most common used scoring function is perhaps the $score$ of variables.

Definition 1 (Score in DLS) *In dynamic local search algorithms, the score of a variable x, denoted by $score(x)$, is the increase of total weight of satisfied clauses caused by flipping x.*

Definition 2 (Decreasing variable) *A variable x is decreasing iff $score(x) > 0$ (as its flip would decrease the cost).*

Although the $score$ of variables have been widely used in local search for SAT and non-partial MaxSAT, its use in (W)PMS is not straightforward, due to the distinction between hard and soft clauses. Recent local search algorithms for W(PMS) also utilize the notion of hard score and soft score, which is the version of score limited to hard and soft clauses respectively.

3 Main Ideas

In this section, we present the two main new ideas in our algorithm.

3.1 New Weighting Mechanism

In this subsection, we propose a new clause weighting scheme for (W)PMS, which works on both hard and soft clauses

while at the same time takes into account the distinction between hard clauses and soft clauses. This is essentially different from previous clause weighting schemes for (W)PMS, which either do not distinguish hard and soft clauses [Thornton *et al.*, 2002; Luo *et al.*, 2015] or just works on hard clauses [Cai *et al.*, 2014; Luo *et al.*, 2017].

Our clause weighting scheme is named **Weighting-PMS**, and works as follows. For each hard clause, we associate an integer number as its weight which is initialized to 1; for each soft clause, we use the original weight (which is 1 for PMS, and is the original weight from the input file for WPMS) as its initial weight. Whenever a "stuck" situation is observed, that is, we cannot decrease the cost by flipping any variable, then clause weights are updated as follows.

- with probability $1 - sp$: for each falsified hard clause c, $w(c) := w(c) + h\_inc$; for each falsified soft clause c, $w(c) := w(c) + 1$ if $w(c) < \zeta$.
- with probability sp (smoothing probability): for each satisfied hard clause c s.t. $w(c) > 1$, $w(c) := w(c) - h\_inc$; for each satisfied soft clause c s.t. $w(c) > 1$, $w(c) := w(c) - 1$.

Our weighting scheme is the first one that works on both hard and soft clauses while at the same time distinguishes hard and soft clauses. The way for updating clause weights are similar to PAWS [Thornton *et al.*, 2004]. Some discussions on the ideas are presented below.

Insights on heavy weighting for hard clause are given below. It is natural to perform clauses weighting for hard clauses, as all hard clauses must be satisfied, and clause weighting helps to identify those difficult hard clauses that are usually falsified in local optima so that the algorithm prefer to satisfy them. Indeed, the intuition here is similar to clause weighting for SAT.

Further, by giving a higher weight increment ($h\_inc \geq 1$) to falsified hard clauses than soft ones, it helps to highlight the hard clauses. In this way, the algorithm prefers to satisfy hard clauses, which is the primary objective of the problem.

Insights on light weighting for soft clause are given below. One can view all soft clauses as a non-partial formula, and by updating weights for soft clauses, we try to satisfy as many soft clauses as possible. Indeed, this is what local search for non-partial MaxSAT usually does [Hoos and Stützle, 2005]. Recent specialized local search algorithms for (W)PMS [Cai *et al.*, 2014; Luo *et al.*, 2017] (Dist and CCEHC) do not update clause weights for soft clauses, as they concern about the possible misleading by weighting for soft clauses.

In our mechanism, we avoid such misleading in two ways. We set a higher weight increment to hard clauses, while the increment is set as 1 for soft clauses. Also, we set a limit on the maximum value of soft clause weight. As is usually the case, there are some soft clauses falsified under optimal assignments. If the weights of soft clauses can grow arbitrarily, such soft clauses would have very high weight, and would prevent us from finding feasible solutions. To avoid this situation, giving a control to the maximum value of soft clause weights is necessary.

3.2 Variable Selection Heuristic

This subsection proposes a novel variable selection heuristic. We design a new variable selection heuristic, which adopts a two-mode dynamic local search framework and exploits the double-make variables. In details, the variable to be flipped is picked as follows:

Firstly, if there exist decreasing variables, the algorithm picks one of the decreasing variables. Observe that for some instances, the number of candidate variables here might be very large, so we use a probabilistic sampling strategy named "best from multiple selections" (BMS) [Cai, 2015] to pick a decreasing variable with high score. Specifically, the algorithm chooses t random variables (with replacement), and returns the one with the highest score. This BMS strategy has proved effective for improving local search on large scale instances, in that it can return an element with very good quality with a constant number of samples. For example, according to calculations in [Cai, 2015], 50 samples guarantees that the returned variable's score is at 10% top among all decreasing variables.

Secondly, if there are no decreasing variables: This means the search gets stuck, as no improving flip is available. In this case, the clause weights are updated, and then a variable in a falsified clause is picked. Since all hard clauses must be satisfied, we select a random falsified hard clause if any, and otherwise a random falsified soft clause will be selected. After the falsified clause is selected, the one with the greatest score is picked. Ties are broken by preferring the variable that is least recently flipped.

4 The SATLike Algorithm

In this section, we present our local search algorithm for (W)PMS, which uses similar techniques in local search for SAT, and thus is called SATLike.

In the beginning, SATLike generates a complete assignment α, and the best found solution α^* is initialized as $\emptyset$. We use the initialization procedure based on unit propagation [Cai *et al.*, 2017] to produce an initial complete assignment for PMS, while the UP-based initialization is not useful for SATLike on WPMS and so the initial assignment is generated randomly for WPMS.

After that, a loop (lines 3-15) is executed to iteratively modify α until a given time limit is reached. During the search, whenever a better feasible solution is found, the best feasible solution α^*, and $cost^*$, are updated accordingly (line 4). At each iteration, SATLike chooses a variables and flips it, according to the variable selection heuristic introduced in the previous section.

If the set of the decreasing variables D is not empty, SATLike picks a decreasing variable with the greatest score from t samples with replacement from D, breaking ties by preferring the one that is least recently flipped.

If there is no decreasing variable, which means the search get stuck, then SATLike updates clause weights according to the Weighting-PMS scheme described in the preceding section, and picks a variable from a falsified clause. Specifically, it chooses a clause randomly from falsified hard clauses if

Algorithm 1: SATLike

Input: PMS instance F, *cutoff*
Output: A feasible assignment α of F and its cost, or "no feasible assignment found"

```
1  begin
2      α := an initial complete assignment; α* := ∅;
3      while elapsed time < cutoff do
4          if ∄ falsified hard clauses & cost(α) < cost* then α* := α; cost* := cost(α);
5          if D := {x|score(x) > 0} ≠ ∅ then
6              v := a variable in D picked by BMS strategy;
7          else
8              update weights of clauses by Weighting-PMS;
9              if ∃ falsified hard cluases then
10                 c := a random falsified hard clause;
11             else c := a random falsified soft clause;
12             v :=the variable with highest score in c;
13         α := α with v flipped;
14     if α* is feasible then return (cost*, α*);
15     else return "no feasible assignment found";
```

any, and from falsified soft clauses otherwise. Then, SATLike picks the variable with the greatest score in the clause.

Finally, when the loop terminates upon reaching the time limit, SATLike returns $cost^*$ and α^* if α^* is a feasible solution. Otherwise, it returns "no feasible assignment found".

5 Experimental Evaluation of SATLike

We evaluate SATlike on (W)PMS benchmarks from the MaxSAT Evaluation (MSE) 2016 and 2017, compared with state of the art local search and SAT-based solvers.

5.1 Experiment Preliminaries

We evaluate SATLike on all benchmarks of PMS and WPMS problems from the MaxSAT Evaluations (MSE) 2016 and 2017. In MSE 2016, there are 3 PMS benchmarks and 3 WPMS benchmarks, for random, crafted and industrial categories respectively, while MSE 2017 merged all PMS categories into a benchmark (PMS_2017), and all WPMS categories into a benchmark (WPMS_2017).

SATLike is implemented in C++ and compiled by g++ with '-O3' option. There are 4 parameters in the SATLike solver:

- t: the number of samplings, used in the BMS heuristic
- sp: the smooth probability, used in Weighting-PMS
- h_inc: the increment for each falsified clause each time, used in Weighting-PMS
- ζ: the limit on maximum value on soft clause weight, used in Weighting-PMS

The parameters are tuned according to our experience, and are listed as followed:

For PMS_Random and PMS_Crafted benchmark: t=15,sp=0.01,h_inc=2,ζ=1;

For PMS_Industrial and PMS_2017 benchmarks: t=42,sp=0.000003,h_inc=1,ζ=400;

For WPMS_Random and WPMS_Crafted benchmark: sp=0.01;

For WPMS_Industrial and WPMS_2017 benchmarks: sp=0.0001;

For all WPMS benchmarks: t=15; h_inc=300 and ζ=500 iff soft clause average weight $>$ 10000, otherwise h_inc=3 and ζ=1;

We compare SATLike with local search solvers. For PMS benchmarks, we choose the local search solver Dist (version 2016) and its latest improved version DeciDist [Cai *et al.*, 2017]. For WPMS benchmarks, we choose the local search solver CCEHC [Luo *et al.*, 2017] and its latest improved version DeciCCEHC [Cai *et al.*, 2017]. DeciDist and DeciCCEHC are the best local search solvers for PMS and WPMS respectively. Also note that the parameters of these local search solvers have been tuned for each (W)PMS benchmark from MSE 2016, and we use these parameter settings for each benchmark as reported in their papers [Cai *et al.*, 2016; Luo *et al.*, 2017].

We compare with three state of the art complete solvers. WPM3 is a famous SAT-based solver and was ranked first in both complete and incomplete tracks in recent MaxSAT Evaluations, including the crafted and industrial categories of PMS and WPMS. Open-WBO [Martins *et al.*, 2014b] is also a SAT-based complete solver, and won the PMS crafted category of MSE 2016 and unweighted complete track of MSE 2017. MaxHS [Davies, 2013] is a SAT-based solver and won the WPMS crafted and industrial categories of MSE 2016 and weighted complete track of MSE 2017. For our experiments, we use the version of WPM3 submitted to incomplete track of MSE 2016 and 2017 (WPM3-in), and the latest release version online for Open-WBO (version 2017.10) and MaxHS (2016). WPM3-in and OpenWBO refine upper bounds during the search, and once a better solution is found, it is printed. But MaxHS only prints the result when it finishes, so the results of MaxHS are only for reference.

Our experiments were conducted on a workstation using an Intel(R) Core(TM) i7-4710MQ CPU with 2.50GHz, 4MB L3 cache and 16 GB RAM, running Ubuntu 14.04 Linux operation system. Each solver is executed once on each instance within 300 seconds (as in MSEs). In each run, the solver prints successively the best solution it has found so far. For each solver on each instance family, we report the number of instances where the solver finds the best solution among all solvers in the table, denoted by "#win." and the mean time of doing so over such "winning" instances (not including the proving time for complete solvers). The number of instances of each family is specified in the column "#ins.". The rules at the MaxSAT Evaluations establish that the winner is the solver which finds the best solution for the most instances and ties are broken by selecting the solver with the minimum mean time. In **bold** we present the best results for each family.

5.2 SATLike vs. Local Search Solvers

In this subsection, we compare SATLike with state of the art local search solvers. We firstly compare SATLike with Dis-

Benchmark	#inst.	*SATLike*		*Dist*		*DeciDist*	
		#win.	time	#win.	time	#win.	time
PMS_Random	210	209	**0.41**	209	1.37	209	2.22
PMS_Crafted	678	**546**	33.78	490	33.01	463	17.30
PMS_Industrial	601	**487**	64.52	156	35.54	209	38.77
PMS_2017	194	**126**	73.02	59	59.48	55	36.80

Table 1: Experimental results of *SATLike* and state of the art local search solvers on PMS benchmarks.

Benchmark	#inst.	*SATLike*		*CCEHC*		*DeciCCEHC*	
		#win.	time	#win.	time	#win.	time
WPMS_Random	502	501	**0.26**	501	1.23	501	1.11
WPMS_Crafted	331	**260**	46.81	208	15.70	202	9.99
WPMS_Industrial	630	**525**	83.16	101	72.76	56	44.16
WPMS_2017	156	**105**	114.00	45	59.10	23	28.64

Table 2: Experimental results of *SATLike* and state of the art local search solvers on WPMS benchmarks.

t and DeciDist on PMS benchmarks. The results are summarized in Table 1. SATLike is the best solver on all the PMS benchmarks, and it significantly outperforms Dist and DeciDist on crafted and industrial PMS benchmarks. The number of winning instances for SATlike is about 1.1 and 2.3 times that for DeciDist on the crafted and industrial PMS benchmark respectively.

Experiment results on comparing SATLike with state of the art local search solvers for WPMS (namely CCEHC and DeciCCEHC) are presented in Table 2. SATLike outperforms CCEHC and DeciCCEHC on all WPMS benchmarks, and obtains remarkable improvements over them on crafted and industrial benchmarks. In particular, for the WPMS industrial benchmark, the winning instances for SATLike is 525, compared to 101 and 56 for CCEHC and DeciCCEHC, which indicates a dramatic improvement.

5.3 SATLike vs. Complete Solvers

We also compare SATLike with three state of the art complete solvers. In Table 3 we summarize the experiment results on comparing SATLike and complete solvers on all benchmarks, and we have the following observations.

SATLike is significantly better than complete solvers on random benchmarks. For crafted benchmarks, SATLike and complete solvers are competitive with each other: complete solvers are better on PMS crafted benchmark while SATLike are better on WPMS crafted benchmark.

For industrial benchmarks, SATLike cannot compete with the complete solvers yet. Nevertheless, the results are still encouraging. The number of winning instances for SATLike achieves about 66% and 72% that for the best complete solver for PMS and WPMS industrial benchmarks respectively, which is a gap that previous local search solvers can hardly reach.

We are also encouraging to see that, for 2017 benchmarks, SATLike is much better than all the complete solvers. These results show the overall strong performance of SATLike on (W)PMS benchmarks.

To gain a more detailed comparison between SATLike and the complete solvers on industrial benchmarks, we present the results on each instance class, as shown in Table 5 and 6. The results show that, SATLike is complementary to the complete solvers on these industrial benchmarks.

Benchmark	#inst.	*SATLike*		*SATLike_alt1*		*SATLike_alt2*	
		#win.	time	#win.	time	#win.	time
PMS_Random	210	**209**	0.41	43	43.50	**209**	**0.12**
PMS_Crafted	678	**554**	34.66	164	75.04	536	32.91
PMS_Industr.	601	**458**	61.37	51	40.57	302	48.84
PMS_2017	194	**115**	90.13	39	27.09	86	63.54
WPMS_Random	502	**501**	0.26	56	56.47	55	52.82
WPMS_Crafted	331	**285**	25.32	96	10.82	273	24.86
WPMS_Industr.	630	**405**	71.59	148	18.84	321	79.54
WPMS_2017	156	**68**	97.51	17	33.14	65	100.66

Table 4: Experimental results of *SATLike* and its alternative versions on all benchmarks

5.4 Experimental Analysis on SATLike

To show the effectiveness of the clause weighting scheme in SATLike, we test two alternatives of SATLike, which replace the Weghting-PMS scheme with its variants. The first one SATLike_alt1 uses the clause weighting scheme without threshold of soft clauses weight. The second one SATLike_alt2 adds the same increment (that is one) for falsified hard and soft clauses. Experiment results show that SATLike significantly outperforms these two alternatives on all the benchmarks (Table 4). This demonstrates the Weighting-PMS scheme in this work plays a key role in SATLike.

6 Conclusions and Future Work

In this work, we proposed to solve the (weighted) partial MaxSAT problem by dynamic local search techniques from SAT background. The strong results show that local search techniques for SAT can be useful to solve (W)PMS problems. Our main idea is a novel clause weighting scheme that works differently for hard clauses and soft clauses. The resulting solver SATLike dramatically outperforms previous local search algorithms on PMS and WPMS benchmarks. Further, when compared to state of the art complete MaxSAT solvers, SATLike is complementary and has better performance on the benchmarks from MSE 2017. Our work significantly narrows the gap between the performance of local search and the currently best approach on the industrial benchmarks.

This work opened a new local search direction for (W)PMS, which is promising to efficiently solve (W)PMS structured instances. We would like to use more advanced local search methods in SAT to enhance our algorithm.

Acknowledgement

This work is supported by National Natural Science Foundation of China 61502464, China National 973 Program 2014CB340301. Shaowei Cai is also supported by Youth Innovation Promotion Association, Chinese Academy of Sciences.

Benchmark	#inst.	*SATLike*		*Open-WBO*		*WPM3*		*MaxHS*	
		#win.	time	#win.	time	#win.	time	#win.	time
PMS_Random	210	**209**	0.41	9	135.09	39	81.27	0	0
PMS_Crafted	678	438	12.28	522	25.95	**553**	16.04	502	30.86
PMS_Industrial	601	340	40.09	**516**	36.88	500	20.23	354	41.98
PMS_2017	194	**108**	67.01	71	137.21	15	116.46	31	231.90
WPMS_Random	502	**501**	0.26	3	79.99	12	86.19	0	0
WPMS_Crafted	331	**233**	34.29	88	37.49	146	22.67	152	38.02
WPMS_Industrial	630	305	55.55	287	34.06	**426**	20.38	246	53.76
WPMS_2017	156	**60**	89.89	45	120.98	29	103.63	27	169.94

Table 3: Experimental results of *SATLike* and state of the art complete solvers on all benchmarks.

Benchmark	#inst.	*SATLike*		*Open-WBO*		*WPM3*		*MaxHS*	
		#win.	time	#win.	time	#win.	time	#win.	time
aes	7	**5**	100.77	1	13.26	1	0.00	4	28.87
atcoss-mesat	18	0	0.00	**14**	178.43	8	124.00	12	67.81
atcoss-sugar	19	0	0.00	**18**	103.80	11	54.90	12	49.76
bcp-fir	32	23	36.26	30	8.80	**31**	3.13	0	0.00
bcp-hipp-yRa1-simp	10	10	23.24	10	**3.73**	9	9.16	7	31.33
bcp-hipp-yRa1-su	38	29	36.67	**38**	8.36	31	2.58	34	30.20
bcp-msp	40	**27**	30.55	26	28.12	20	24.54	22	113.67
bcp-mtg	30	22	11.90	30	**0.03**	30	0.09	30	1.14
bcp-syn	38	**29**	32.51	11	72.95	20	17.37	21	3.24
circuit-trace-compacton	4	0	0.00	4	5.09	4	**2.90**	4	80.58
close_solutions	50	43	16.90	28	95.51	**49**	36.98	30	119.60
des	50	30	83.52	**48**	79.66	40	56.83	4	40.02
haplotype-assembly	6	2	138.45	5	49.75	5	**0.19**	1	4.89
hs-timetabling	2	0	0.00	1	**4.44**	1	15.68	1	10.99
mbd	46	2	194.01	41	53.70	**44**	11.29	27	68.06
packup-pms	40	38	35.85	40	8.38	**40**	1.67	7	12.70
pbo-nencdr	25	3	172.67	25	24.49	25	**19.70**	25	40.73
pbo-nlogencdr	25	23	50.85	25	5.14	25	**2.02**	25	11.71
pbo-routing	15	15	0.58	15	1.23	15	**0.33**	9	4.07
protein_ins	12	12	123.87	12	**29.83**	12	37.15	2	15.90
tpr-Multiple_path	36	0	0.00	**36**	31.82	31	37.55	31	53.00
tpr-One_path	25	1	229.95	25	2.81	25	**2.23**	25	3.97
treewidth-computation	33	26	11.34	**33**	25.96	23	23.06	21	16.70
total	601	340	40.09	516	36.88	500	20.23	354	41.98

Table 5: Detailed comparison of *SATLike* and complete solvers on PMS Industrial benchmark

Benchmark	#inst.	*SATLike*		*Open-WBO*		*WPM3*		*MaxHS*	
		#win.	time	#win.	time	#win.	time	#win.	time
abstraction-refinement	11	2	243.75	2	270.08	**7**	82.87	0	0.00
BTBNSL	60	0	0.00	3	252.99	19	119.14	**42**	178.69
correlation-clustering	129	**101**	124.97	0	0.00	14	73.04	21	150.86
haplotyping-pedigrees	100	38	32.21	**100**	13.14	97	9.73	26	12.08
hs-timetabling	14	0	0.00	7	**85.18**	7	100.31	0	0.00
packup-wpms	99	19	13.10	0	0.00	**99**	9.64	7	0.00
preference_planning	29	12	69.61	29	**2.60**	29	6.75	29	30.93
railway-transport	11	0	0.00	**5**	165.32	4	29.64	4	189.75
relational-inference	9	1	0.42	1	29.27	4	94.39	**6**	20.89
timetabling	26	0	0.00	**15**	119.25	13	57.83	5	72.13
upgradeability-problem	100	97	5.15	92	25.78	**100**	2.16	90	1.05
wcsp-dir	21	**19**	44.82	16	25.12	17	16.30	8	0.00
wcsp-log	21	16	10.79	**17**	63.12	16	17.31	8	0.07
total	630	305	55.55	287	34.06	426	20.38	246	53.76

Table 6: Detailed comparison of *SATLike* and complete solvers on WPMS Industrial benchmark

References

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

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

[Bacchus *et al.*, 2014] Fahiem Bacchus, Jessica Davies, Maria Tsimpoukelli, and George Katsirelos. Relaxation search: A simple way of managing optional clauses. In *Proceedings of AAAI-14*, pages 835–841, 2014.

[Cai *et al.*, 2014] Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial MaxSAT. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI), Québec City, Québec, Canada*, pages 2623–2629, 2014.

[Cai *et al.*, 2016] Shaowei Cai, Chuan Luo, Jinkun Lin, and Kaile Su. New local search methods for partial maxsat. *Artificial Intelligence*, 240:1–18, 2016.

[Cai *et al.*, 2017] Shaowei Cai, Chuan Luo, and Haochen Zhang. From decimation to local search and back: A new approach to maxsat. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 571–577, 2017.

[Cai, 2015] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina*, pages 747–753, 2015.

[Cha *et al.*, 1997] Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, and Shuichi Miyazaki. Local search algorithms for partial MAXSAT. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), Providence, Rhode Island*, pages 263–268, 1997.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proceedings of the 17th International Conference of Principles and Practice of Constraint Programming (CP), Perugia, Italy*, pages 225–239, 2011.

[Davies, 2013] Jessica Davies. Solving MaxSAT by decoupling optimization and satisfaction, 2013. PhD thesis.

[Fu and Malik, 2006] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proceedings of the 9th International Conference of Theory and Applications of Satisfiability Testing (SAT), Seattle, WA, USA*, pages 252–265, 2006.

[Grégoire *et al.*, 2014] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure. An experimentally efficient method for (mss, comss) partitioning. In *Proceedings of AAAI-14*, pages 2666–2673, 2014.

[Hoos and Stützle, 2005] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2005.

[Luo *et al.*, 2015] Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. CCLS: An efficient local search algorithm for weighted maximum satisfiability. *IEEE Trans. on Computers*, 64(7):1830–1843, 2015.

[Luo *et al.*, 2017] Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. *Artif. Intell.*, 243:26–44, 2017.

[Marques-Silva *et al.*, 2013] João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. On computing minimal correction subsets. In *Proceedings of IJCAI-13*, 2013.

[Martins *et al.*, 2014a] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for maxsat. In *Proceedings of the 20th International Conference of Principles and Practice of Constraint Programming (CP), Lyon, France*, pages 531–548, 2014.

[Martins *et al.*, 2014b] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of the 17th International Conference of Theory and Applications of Satisfiability Testing, Vienna, Austria*, pages 438–445, 2014.

[Mencía *et al.*, 2015] Carlos Mencía, Alessandro Previti, and João Marques-Silva. Literal-based MCS extraction. In *Proceedings of IJCAI-15*, pages 1973–1979, 2015.

[Morgado *et al.*, 2013] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and Joao Marques-Silva. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[Narodytska and Bacchus, 2014] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI), Québec City, Québec, Canada*, pages 2717–2723, 2014.

[Thornton and Sattar, 1998] John Thornton and Abdul Sattar. Dynamic constraint weighting for over-constrained problems. In *Proceedings of the 5th Pacific Rim International Conference on Artificial Intelligence (PRICAI), Singapore*, pages 377–388, 1998.

[Thornton *et al.*, 2002] John Thornton, Stuart Bain, Abdul Sattar, and Duc Nghia Pham. A two level local search for MAX-SAT problems with hard and soft constraints. In *Proceedings of the 15th Australian Joint Conference on Artificial Intelligence (AI), Canberra, Australia*, pages 603–614, 2002.

[Thornton *et al.*, 2004] John Thornton, Duc Nghia Pham, Stuart Bain, and Valnir Ferreira Jr. Additive versus multiplicative clause weighting for SAT. In *Proceedings of the 19th National Conference on Artificial Intelligence, (AAAI), San Jose, California, USA*, pages 191–196. AAAI Press / The MIT Press, 2004.