Artificial Intelligence 240 (2016) 1–18

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

New local search methods for partial MaxSAT

Shaowei Cai [a,*], Chuan Luo [b,c], Jinkun Lin [c], Kaile Su [d]

[a] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[b] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[c] School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
[d] Institute for Integrated and Intelligent Systems, Griffith University, Brisbane 4111, Australia

ARTICLE INFO

Article history:
Received 27 June 2015
Received in revised form 29 April 2016
Accepted 28 July 2016
Available online 3 August 2016

Keywords:
Partial MaxSAT
Local search
Hard and soft score
Initialization

ABSTRACT

Maximum Satisfiability (MaxSAT) is the optimization version of the Satisfiability (SAT) problem. Partial Maximum Satisfiability (PMS) is a generalization of MaxSAT which involves hard and soft clauses and has important real world applications. Local search is a popular approach to solving SAT and MaxSAT and has witnessed great success in these two problems. However, unfortunately, local search algorithms for PMS do not benefit much from local search techniques for SAT and MaxSAT, mainly due to the fact that it contains both hard and soft clauses. This feature makes it more challenging to design efficient local search algorithms for PMS, which is likely the reason of the stagnation of this direction in more than one decade.

In this paper, we propose a number of new ideas for local search for PMS, which mainly rely on the distinction between hard and soft clauses. The first three ideas, including weighting for hard clauses, separating hard and soft score, and a variable selection heuristic based on hard and soft score, are used to develop a local search algorithm for PMS called *Dist*. The fourth idea, which uses unit propagation with priority on hard unit clauses to generate the initial assignment, is used to improve *Dist* on industrial instances, leading to the *DistUP* algorithm.

The effectiveness of our solvers and ideas is illustrated through experimental evaluations on all PMS benchmarks from the MaxSAT Evaluation 2014. According to our experimental results, *Dist* shows a significant improvement over previous local search solvers on all benchmarks. We also compare our solvers with state-of-the-art complete PMS solvers and a state-of-the-art portfolio solver, and the results show that our solvers have better performance in random and crafted instances but worse in industrial instances. The good performance of *Dist* has also been confirmed by the fact that *Dist* won all random and crafted categories of PMS and Weighted PMS in the incomplete solvers track of the MaxSAT Evaluation 2014.

* Corresponding author.
E-mail address: shaoweicai.cs@gmail.com (S. Cai).

http://dx.doi.org/10.1016/j.artint.2016.07.006
0004-3702/

1. Introduction

1.1. The problem

The Maximum Satisfiability problem (MaxSAT) is the optimization version of the Satisfiability problem (SAT). Given a propositional formula in the conjunctive normal form (CNF), i.e., $F = \bigwedge_i \vee_j l_{ij}$, the task in MaxSAT is to find an assignment to the variables that maximizes the number of satisfied clauses. A significant generalization of MaxSAT is the Partial MaxSAT (PMS) problem, in which clauses are divided into hard and soft clauses and the goal is to find an assignment that satisfies all hard clauses and maximizes the number of satisfied soft clauses. PMS is particularly interesting from an algorithmic point of view because the algorithms can exploit the distinction between hard and soft constraints. Such a structural feature has a great impact on the performance of algorithms.

Combinatorial optimization problems containing hard and soft constraints are very common in real world situations. PMS allows to encode such problems in a more natural and compact way than SAT and MaxSAT. PMS solvers has been successfully used in many fields, including network routing [31], scheduling problems [50] and timetabling problems [15]. More recent applications of PMS include FPGA routing [19], the haplotype inference by pure parsimony (HIPP) problem seeking to explain the genetic makeup of a population [20], as well as various planning problems, all of which are reviewed in a PhD thesis [18]. Some application problems, such as the optimal protein alignment problem and the HIPP problem, have been encoded into PMS and used in the MaxSAT evaluations [7].

1.2. Related work

PMS (as a generalization of MaxSAT) is an NP-hard problem. Most existing practical algorithms for PMS are complete search algorithms, which prove the optimality of the solutions they find when they terminate (before reaching a time limit). There are numerous complete algorithms for solving PMS. A large family of complete algorithms for PMS employ a branch and bound algorithm strategy; they usually incorporate lower bound computation methods and utilize inference rules [35, 37,23,36,17].

In the last decade, the use of SAT solvers for solving MaxSAT problems has emerged as another paradigm. This approach is usually referred to as the SAT-based approach, and it is based on iteratively calling a SAT solver. SAT-based MaxSAT solvers can be divided into two categories: core-guided and model-guided. Core-guided algorithms refine the lower bound and guide the search with unsatisfiable subproblems (cores), while model-guided algorithms refine the upper bound and guide the search with satisfying assignments (models). Since Fu and Malik described their core-guided PMS algorithm PM1 [19], there has been much interest in this direction, and many core-guided solvers have been developed [9,16,33,5,43,41, 40,44,45]. Core-guided PMS solvers are especially well known for their good performance on industrial instances. Some other approaches reduce PMS into a well-known optimization problem and use an off-the-shelf solver for such a problem. A successful example of such approaches is to reduce PMS into integer linear programs (ILP) and solve the instance by a Mixed Integer Programming (MIP) solver [3,48].

Although complete algorithms have shown great success in PMS solving, they may fail to find a good solution within reasonable time for large instances, as essentially they are systemic search approaches that explore the whole search space. An alternative approach to tackling the PMS problem is local search. As a popular approach to solving NP-hard combinatorial problems, local search is well known for its ability to quickly find a good-quality approximate (sometimes even optimal) solution. For combinatorial optimization problems, local search algorithms typically maintain a complete assignment for the problem, and iteratively modify the assignment (in the case of MaxSAT and PMS, this means flipping the value of a variable). An important feature of local search algorithms is that they keep track of the best assignment that was found throughout the search. This makes them anytime algorithms [55], i.e., they are expected to find better and better solutions the more time it keeps running. Therefore, efficient local search algorithms are particularly useful in real world applications where approximate solutions are acceptable while time limit is short or time resource is very important.

Local search has been shown to be effective for solving SAT, and is among the best known methods currently available for solving certain types of SAT instances, particularly since recent progress due to several algorithms [8,34,10,13]. Local search techniques for SAT can be directly applied or easily adapted to MaxSAT. Most early successful local search algorithms for SAT have been extended for approximating MaxSAT in the UBCSAT system [53], and a survey can be found in [25]. There have been also efforts devoted to specialized local search algorithms for MaxSAT, e.g., [49,1,54,21]. In particular, a recent local search algorithm for weighted MaxSAT called *CCLS* [38] won four categories in the incomplete track of the MaxSAT Evaluation 2013, thanks to the configuration checking strategy, which was initially proposed in [11] and has shown success in SAT solving [10]. Local search algorithms for weighted MaxSAT can be used to solve PMS, as PMS can be encoded into weighted MaxSAT, by setting the weight of each soft clause as 1 and that of each hard clause as the number of soft clauses plus 1. However, local search MaxSAT solvers cannot achieve comparable performance with complete PMS solvers on structured PMS instances, as witnessed by recent MaxSAT Evaluations.

Compared to the great success of local search for SAT and MaxSAT, there are only few studies on local search for PMS [31, 15,50,51], which were proposed since more than a decade. Although many local search techniques for SAT are also effective for solving MaxSAT, local search algorithms for PMS do not benefit that much from the techniques for SAT, mainly due to the fact that it contains both hard and soft clauses. To develop effective local search algorithms for PMS, it is necessary to

exploit the distinction between hard and soft clauses. One of the earliest works in this line is a weighted version of *WalkSAT* [31], which also tackles PMS as weighted MaxSAT, but prefers to flip variables in falsified hard clauses. Afterwards, Cha et al. observed that the larger the weight differential between hard clauses and soft clauses, the slower the search is [15]. This insight has led to an algorithm in which the hard clause weight is set to a hand-tuned optimal level (rather than simply set to the number of soft clauses plus 1) [15]. This was further improved by Thornton et al. by maintaining a dynamic weight differential between hard and soft clauses [50,51], resulting in the *TWO-LEVEL* algorithm. Experimental results in [50] showed the superiority of dynamic weighting strategies over the fixed weighting strategy in [15].

1.3. Main contributions

In this paper, we propose a number of new ideas for local search for PMS, which mainly rely on the distinction between hard and soft clauses. Based on these ideas, we develop two local search algorithms for PMS called *Dist* and *DistUP*, the latter of which is an improved version of the former for industrial PMS instances.

The first idea we propose is a clause weighting scheme that only works on hard clauses. Indeed, the hard clauses of a PMS instance are tackled as a SAT formula for which weighted local search algorithms are efficient [52,46,10]. With the diversification achieved by this weighting scheme, the algorithm tends to visit different satisfying assignments for hard clauses, and thus different groups of feasible assignments. In this way, the algorithm can better explore the space of feasible assignments, and thus is more likely to come across better feasible assignments.

The second and more important idea is to separate hard score and soft score. Here the hard score of a variable is the change on the number (or total weight) of satisfied hard clauses caused by flipping the variable, and the soft score of a variable is the change on the number (or total weight) of satisfied soft clauses caused by flipping the variable. By separating hard score and soft score, the algorithm becomes more flexible, in the sense that it can pick the flipping variable according to either hard score or soft score, or both, according to different situations.

The third idea is a variable selection heuristic based on hard score and soft score. The heuristic distinguishes three different situations during the search, and uses hard score and soft score in different ways under each situation.

The three ideas mentioned above are used in developing a local search algorithm for PMS dubbed *Dist*, as it makes effective use of *Dist*inctions between hard and soft clauses. Results of the MaxSAT Evaluation 2014 as well as our experiments show that *Dist* significantly outperforms previous local search solvers on all benchmarks from the MaxSAT Evaluation 2014, with a remarkable improvement in terms of the number of "winning" instances on structured PMS benchmarks. We also compare *Dist* with latest state-of-the-art complete solvers and a state-of-the-art portfolio solver on PMS benchmarks from the MaxSAT Evaluation 2014. Experimental results show that *Dist* outperforms the complete solvers on random and crafted benchmarks, while its performance on industrial instances is still considerably worse than complete solvers.

The aforementioned ideas and the *Dist* algorithm have been presented in [12], but in this article we add more experiments and replace the complete solvers in our experiments with latest state-of-the-art ones. The following contributions are new in this article.

In order to improve the performance of *Dist* on industrial PMS instances, we propose an initialization procedure called *PrioUP*, which utilizes unit propagation and puts priority on hard unit clauses. The procedure produces a complete assignment, which is then used as the initial assignment for the *Dist* solver. The resulting solver is called *DistUP*, and it significantly improves *Dist* on industrial instances, although it still cannot rival complete solvers.

We also perform experimental analysis and additional investigations on the ideas in this work. In detail, we compare *Dist* with its four alternative versions, and the experimental results illustrate the effectiveness of the ideas; more interestingly, all alternatives based on separation of hard and soft score have better performance than previous local search algorithms, indicating separation of hard and soft score is an essential technique and opens up a new direction for local search algorithms for PMS (and also weighted PMS). We also study the effectiveness of the *PrioUP* procedure on Weighted PMS industrial instances, and provide a discussion on the initialization procedure.

1.4. Structure of the paper

The remainder of this paper is organized as follows: some preliminary concepts are given in next section. We present in detail three new local search ideas for PMS in Section 3, and present the *Dist* algorithm in Section 4. Then we present the experimental study on *Dist* in Section 5. After that, we propose the *PrioUP* procedure, and apply it to improve *Dist* in Section 6, where we also present experiments on the improved algorithm *DistUP* and a discussion on the initialization procedure. Finally, we give some concluding remarks and directions for future research.

2. Preliminaries

Given a set of n Boolean *variables* $\{x_1, x_2, ..., x_n\}$, a *literal* is either a variable x_i (which is called a positive literal) or its negation $\neg x_i$ (which is called a negative literal). A *clause* is a disjunction of literals (i.e., $C_i = \ell_{i1} \vee \ell_{i2} \vee ... \vee \ell_{ij}$). A conjunctive normal form (CNF) formula $F = C_1 \wedge C_2 \wedge ... \wedge C_m$ is a conjunction of clauses. Alternatively, clauses can be represented as sets of literals and a formula as a multiset of clauses.

The variable to which a literal l refers is denoted by $var(l)$. For a literal l, its polarity, denoted by $polarity(l)$, is 1 if l is positive and 0 if l is negative. For a literal l, we denote by $\neg l$ the literal of opposite polarity, and $\neg\neg x_i = x_i$. A complete truth assignment is a mapping that assigns to each variable either 0 or 1. Given an assignment, a clause is *satisfied* if it has at least one true literal, and *falsified* otherwise.

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 falsified (satisfied) clauses is minimized (maximized). The MaxSAT problem is a special case of PMS in which there are no hard clauses. We can also view the SAT problem as a special case of PMS in which there are no soft clauses.

For a PMS instance F, we say a truth assignment α is *feasible* iff it satisfies all hard clauses in F, and the *cost* of a feasible assignment α, denoted by $cost(\alpha)$, is defined to be the number of falsified soft clauses under α. An optimal assignment is a feasible assignment with minimum cost. The basic schema for local search algorithms for PMS (as with MaxSAT) is as follows. Starting with a complete assignment, the algorithm chooses a variable and flips it (i.e., changes its truth value) in each subsequent step, trying to find a feasible assignment with a lower cost. PMS can be encoded as weighted MaxSAT where hard clauses are associated with a weight larger than the total weight of soft clauses. A common variable property used in local search algorithms for weighted MaxSAT is the *score* property, which is defined as the change on the total weight of satisfied clauses caused by flipping the variable. However, in this work, a hard (resp. soft) score is calculated only on the hard (resp. soft) clauses.

A clause containing only one literal is a *unit clause*. For convenience, we use $polarity(p)$ to denote the polarity of the literal in the unit clause p. The process of conditioning a CNF formula F on a literal l amounts to replacing every occurrence of literal l by the constant true, replacing $\neg l$ by the constant false, and simplifying accordingly. Based on the multi-set notation of a CNF formula, the result of conditioning a CNF formula F on a literal l is denoted by $F|_l$, which is defined as follows: $F|_l = \{c \setminus \{\neg l\} | c \in F, l \notin c, \neg l \in c\} \cup \{c | c \in F, l \notin c, \neg l \notin c\}$, and can be described succinctly as $F|_l = \{c \setminus \{\neg l\} | c \in F, l \notin c\}$. Note that $F|_l$ does not contain any literal l or $\neg l$.

The unit propagation technique is quite simple: For a given CNF formula, we collect all unit clauses in it, and then assume that variables are set to satisfy these unit clauses. That is, if the unit clause $\{x_i\}$ appears in the formula, we set x_i to true; and if the unit clause $\{\neg x_i\}$ appears in the formula, we set x_i to false. We then condition the formula on these settings. The iterative application of this rule until no more unit clause remains is called *unit propagation* (UP).

3. Exploiting the distinction between hard and soft clauses

In this section, we present three new ideas for local search for PMS, which heavily rely on the distinction between hard and soft clauses. These three new ideas are 1) weighting for hard clause, 2) separating hard score and soft score, and 3) a variable selection heuristic based on hard and soft score. These ideas form the major components of the *Dist* algorithm.

3.1. Weighting for hard clauses

In this subsection, we propose a clause weighting scheme that works only on hard clauses. This is essentially different from previous local search algorithms for PMS which also utilize clause weighting schemes, as they increase weights of all falsified clauses, including both hard and soft ones [15,50,51].

We now describe the weighting scheme. For each hard clause, we associate an integer number as its weight, which is initialized to 1 at the start of the algorithm.[1] Whenever a "stuck" situation w.r.t. hard clauses is observed, that is, we cannot decrease the total weight of falsified hard clauses by flipping any variable, then hard clause weights are updated as follows:

with probability sp (smoothing probability), for each satisfied hard clause whose weight is larger than one, the clause weight is decreased by one; otherwise, the clause weights of all falsified hard clauses are increased by one.

The way that hard clause weights are updated is similar to the PAWS scheme [52], and thus we refer to this new weighting scheme as HPAWS (the hard clause version of PAWS). The only difference between PAWS and HPAWS is the condition to decrease clause weights. PAWS increases weights for falsified clauses by one in each step, and all clause weights are decreased by one after a fixed number of increases; while HPAWS employs a probability parameter to decide whether to increase weights or decrease weights in the step.

Our weighting scheme is the first one working only on hard clauses. Some intuitive explanations behind the idea of weighting only for hard clauses are presented below.

Why to use weighting for hard clauses: (1) Clause weighting for hard clauses helps to obtain feasible solutions. It identifies those hard clauses that are usually falsified in local optima, so that the algorithm can prefer to satisfy such hard clauses. In this way, we can avoid the situation that some hard clauses are always falsified in local optima. (2) Moreover, with the diversification achieved by the weighting scheme, the algorithm tends to visit different satisfying assignments for hard clauses, and thus different groups of feasible assignments. In this way, the algorithm can better explore the space of feasible assignments, and thus is more likely to come across better feasible assignments. In summary, using weighting for hard clauses is inspired by the success of weighting techniques for SAT [27,52,46,10].

[1] When PMS is encoded as weighted MaxSAT (as in MaxSAT evaluations), hard clauses have weights due to the encoding. We note that the weights used in our algorithm are independent of the original weights of hard clauses. Actually, we only use the original weights to recognize hard clauses.

Why not to use weighting for soft clauses: Some soft clauses might be usually falsified in local optima, due to the high cost of satisfying them, i.e., making more clauses falsified. Indeed, as is usually the case, there are some soft clauses falsified under optimal assignments. However, the object of PMS is to satisfy as many soft clauses as possible rather than all of them, under the constraint that all hard clauses are satisfied. Therefore, our opinion is that compelling the algorithm to satisfy "difficult" soft clauses at the price of falsifying more soft clauses has no clear benefits, and would mislead the algorithm towards feasible solutions with more falsified soft clauses.

3.2. Separating hard score and soft score

Most local search algorithms for SAT and MaxSAT problems utilize the variable property *score*, which measures the increase of the number (or total weight) of satisfied clauses caused by flipping a variable x. Previous local search algorithms for PMS also utilize the *score* property to pick variables. In this work, however, we propose to separate hard score and soft score, which allows us to make better use of the special structure of PMS and thus design more efficient local search algorithms for PMS.

The concepts of hard score and soft score are formally defined as follows:

Definition 1. (hard score) The hard score of a variable x, denoted by $hscore(x)$, is the increase of the number (or total weight) of satisfied hard clauses caused by flipping x.

For convenience, we say a variable x is a 0-*hscore* variable if and only if $hscore(x) = 0$.

Definition 2. (soft score) The soft score of a variable x, denoted by $sscore(x)$, is the increase of the number (or total weight) of satisfied soft clauses by flipping x.

Dist adopts the weighted version of hard score, as it employs a clause weighting scheme for hard clauses. *Dist* adopts the unweighted version of soft score for PMS instances, and the weighted version of soft score for Weighted PMS instances.

To facilitate our discussions about *Dist* afterwards, we give more definitions here. In local search algorithms for SAT and MaxSAT, a variable is said to be decreasing (which means its flip would decrease the cost of the assignment) if its score is positive, and increasing if its score is negative. Now, we extend the notion of decreasing variables to hard score and soft score.

Definition 3. For a variable x, x is hard-decreasing iff $hscore(x) > 0$, and is soft-decreasing iff $sscore(x) > 0$.

Previous local search algorithms for PMS utilize the score property, which can be seen as a weighted sum of hard score and soft score in the form of $A \cdot hscore(x) + sscore(x)$, where A is a positive integer number. Intuitively, because hard clauses are compelled to be satisfied, the factor A should be set very large (as an extreme case, A is set to the number of soft clauses plus one). On the other hand, the search would be quite restricted if A is too large. Hence, the main concern in previous local search algorithms for PMS is how to control the value of A to make the search more effective [15,50,51].

However, no matter what strategies they use, these algorithms set A to be a relatively large number so that hard clauses are more important than soft ones. Therefore, when using a heuristic preferring variables with greater scores, it is likely that those variables with greater *hscores* are actually picked. This is, in our opinion, a drawback of previous local search algorithms for PMS. We provide an informal explanation for this drawback as follows. Generally speaking, such local search algorithms focus on finding feasible solutions. It is difficult for them to leave an area of a cluster of feasible solutions, as there are infeasible solutions between two clusters of feasible solutions. Thus, the search space they explore would be quite limited.

In contrast to previous local search algorithms, we propose to separate hard score and soft score. This makes the algorithm more flexible, in the sense that it can pick the flipping variable according to either hard score or soft score, or both, according to different situations.

3.3. Variable selection based on hard and soft scores

Based on the separation of hard and soft scores, we propose a variable selection heuristic for PMS that works in three levels. The scenario that the search faces varies considerably, and we divide them into three situations. The variable to be flipped is picked according to different scoring functions under each situation as follows:

1. There exist hard-decreasing variables. Flipping such variables will decrease the total weight of falsified hard clauses, and hence lead the search to feasible solutions. As the preliminary goal of PMS is to find a feasible solution, such variables are given the highest priority of being flipped.
   However, there is still one question that needs to be answered: when there is more than one hard-decreasing variables, which one should be selected? Two natural answers are to pick the best one or to pick one uniformly at random. It

is difficult to decide which variable is the best, unless there is some variable whose *hscore* and *sscore* are both the greatest, which rarely happens. On the other hand, picking a variable at random uniformly is too diversifying in most cases. In our algorithm, we employ a balanced strategy called Best from Multiple Selections (BMS) [14], which chooses t variables (t is an integer parameter) randomly with replacement from the set of hard-decreasing variables and returns the one with the greatest *hscore*, breaking ties by preferring the one with the greatest *sscore*.

2. There are no hard-decreasing variables, yet there are variables with *hscore* of 0. In this case, we further consider those 0-*hscore* variables with positive *sscore*, as flipping such variables would decrease falsified soft clauses without breaking more hard clauses. Therefore, flipping such variables is of clear benefit, especially when no hard-decreasing variables exist. Specifically, a variable with the best positive *sscore* is selected uniformly at random, which seems a fastest way towards the goal of PMS under this situation.
3. There are neither hard-decreasing variables nor 0-*hscore* soft-decreasing variables. This means the search gets stuck, as no improving flip is available. In this case, the weights of hard clauses are updated, and then a variable is picked from a falsified clause. Since hard clauses are compelled to be satisfied, we select a random falsified hard clause if any, and otherwise a random falsified soft clause is selected (this strategy has been used in [31]).
   A question is then which variable should be chosen from the selected clause. Since before the stuck situation occurs we mainly focus on satisfying hard clauses or at least protecting them, here we need to diversify the search w.r.t. variables' *hscores*. A good heuristic is to pick the variable with the greatest *sscore*, which is independent of the variables' *hscores*, while at the same time helps to satisfy as many soft clauses as possible.

Note that, the early version of Dist [12] did not use the BMS strategy. In MaxSAT Evaluation 2014, we adopted the version described in this paper, except small modification for random benchmarks (the performance is similar).

4. The *Dist* algorithm

Based on the ideas from the preceding section, we develop an efficient local search algorithm for solving PMS, which is called *Dist*, as it makes effective use of the *Dist*inctions between hard and soft clauses.

4.1. Description of Dist

The *Dist* algorithm is outlined in Algorithm 1:

Algorithm 1: *Dist*.

```
   Input: PMS instance F, cutoff, parameters wp, t and sp
   Output: A feasible assignment α* of F, or "no feasible assignment found"
1  α := randomly generated truth assignment;
2  cost* := +∞;
3  while elapsed time < cutoff do
4      if ∄ falsified hard clauses & cost(α) < cost* then
5          α* := α; cost* := cost(α);
6      if H := {x|hscore(x) > 0} ≠ ∅ then
7          H' := {variables chosen from H via t samples with replacement};
8          v := the variable with the greatest hscore in H', breaking ties by preferring the one with the greatest sscore;
9      else if S := {x|hscore(x)=0 & sscore(x)> 0} ≠ ∅ then
10         v := a variable in S with the greatest sscore, breaking ties randomly;
11     else
12         update weights of hard clauses according to HPAWS;
13         if ∃ falsified hard clauses then c := a random falsified hard clause;
14         else c := a random falsified soft clause;
15         if with probability wp then v := a random variable in c;
16         else v := a variable in c with the greatest sscore;
17     α := α with v flipped;
18 if cost* ≤ #(soft clauses) then return (cost*, α*);
19 else return "no feasible assignment found";
```

In the beginning, *Dist* generates a complete assignment α randomly, and the cost of the best feasible solution, denoted by $cost^*$, is initialized to $+\infty$. After the initialization, a loop (lines 3–17) is executed to 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 (lines 4–5).

In each iteration, *Dist* flips a variable, which is selected according to the variable selection heuristic mentioned in the previous section. First, if the set of hard-decreasing variables H is not empty, *Dist* picks a hard-decreasing variable with the greatest *hscore* from t samples with replacement from H, breaking ties by preferring the one with the greatest *sscore*

(lines 6–8). If no hard-decreasing variable exists, then *Dist* picks a best variable (w.r.t. *sscore*) from the set S of variables with 0-*hscore* and positive *sscore*, breaking ties randomly (lines 9–10). If both H and S sets are empty, which means the algorithm gets stuck, then *Dist* updates hard clause weights according to the HPAWS weighting scheme described in the preceding section (line 12), and picks a variable from a falsified clause. Specifically, it chooses a clause randomly from falsified hard clauses if any, and from falsified soft clauses otherwise (lines 13–14). In order to make the algorithm more robust, we employ a random walk with a small probability in each random step (line 15), as suggested in [26]; Otherwise, the variable with the greatest *sscore* from the chosen falsified clause is selected (line 16).

Finally, when the loop terminates upon reaching the time limit, *Dist* reports $cost^*$ and the best feasible solution α^* that has been found, if $cost^*$ is not greater than the number of soft clauses (this means a feasible solution is found, since $cost^*$ is initialized as $+\infty$ and is updated only when a better feasible solution is found). Otherwise, *Dist* reports "no feasible assignment found".

5. Experimental evaluation of *Dist*

We empirically evaluate *Dist* on PMS benchmarks from the MaxSAT Evaluation 2014, including all the three categories namely random, crafted and industrial. We compare *Dist* with state-of-the-art local search solvers and the best complete solvers for each benchmark, as well as a state-of-the-art portfolio solver. Finally, we empirically analyze the effectiveness of the underlying ideas in *Dist*.

5.1. Parameter setting and experiment setup

Dist is implemented in C++ and complied using g++ with the '-O3' option. There are three parameters in *Dist*, namely the t parameter for the "Best from Multiple Selections" strategy, wp for random walk, and sp for the weighting scheme. Parameter settings have important impact on the performance of heuristic algorithms. To tune parameters automatically, some automatic configuration tools [29,30] have been developed and have shown their power on solving SAT and mixed integer programming. In this work, we utilized a powerful automatic configuration tool called *SMAC* (Sequential Model-based Algorithm Configuration) [30] to tune the parameter settings for our *Dist* algorithm. The parameter settings suggested by *SMAC* are as follows:

- For random benchmark: $t = 3$, $wp = 0.1$, $sp = 0.01$;
- For crafted benchmark: $t = 17$, $wp = 0.105$, $sp = 0.013$;
- For industrial benchmark: $t = 42$, $wp = 0.091$, $sp = 0.00003$.

As for the optimal setting of parameters, we observed that there were two outlier instance families namely "maxclique/structured" and "reversi", both from the crafted benchmark. We found that, the optimal value of t for "maxclique/structured" and the optimal value of sp for "reversi" instances tend to be much smaller than the optimal values for the rest of the crafted benchmark. And for these two families, we use *SMAC* to find a different setting: (2, 0.1, 0.01) for the "maxclique/structured" family, and (15, 0.1, 0.0001) for the "reversi" family.

Our experiments were conducted on a workstation using an Intel(R) Core(TM) i7-2640M CPU with 2.8 GHz, 4 MB L3 cache and 16 GB RAM, running Ubuntu 12.04 Linux operation system. We follow the evaluation methodology adopted in the incomplete solver track of the MaxSAT evaluations: Each solver is executed once on each instance within a time limit which is set to 300 seconds (the same as in the incomplete solver track of the MaxSAT evaluations). In each run, the solver prints successively the best solution it has found so far. For each solver on each instance family, we report within parentheses the number of instances where the solver finds the best solution, 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. Comparing Dist with local search solvers

We first compare *Dist* with state-of-the-art local search solvers, including those participating in the PMS categories of the incomplete track of the MaxSAT Evaluation 2014, as well as the *TWO-LEVEL* solver [51], which is the best specialized local search solver for PMS we can find, although it did not participate in the MaxSAT Evaluations. Most solvers participating in the PMS categories of the incomplete track are slightly modified from complete solvers, and there are only three local search solvers that participated in the PMS categories, namely *Dist*, *CCLS* and *CCMPA*. This is not surprising, as before *Dist*, there has been little progress in local search solvers for PMS.

Results on Random PMS Benchmark: Table 1 shows the comparative results of local search solvers on the random PMS benchmark in the MaxSAT Evaluation 2014. As is clear from the table, *Dist* gives the best performance on all instance families. We observe that *Dist* is more robust and more efficient than its competitors. *Dist* finds the best solution for all

Table 1
Comparative results for *Dist* and other local search solvers for PMS on random PMS benchmark. One instance from the "pmax2sat/hi" family (file_rpms_wcnf_L2_V150_C4000_H150_ 1.wcnf) has been proved (by most complete solvers in MaxSAT Evaluation 2014) that the hard clauses compose an unsatisfiable formula and thus no feasible assignment exists for the instance.

Instance set	#ins.	*Dist*	*CCLS*	*CCMPA*	*TWO-LEVEL*
min2sat/v160c800l2	30	**4.20(30)**	6.62(30)	33.66(30)	0(0)
min2sat/v260c1040l2	30	**2.86(30)**	3.63(30)	28.98(30)	56.25(2)
min3sat/c70v350l3	30	**1.03(30)**	7.92(29)	10.62(30)	15.94(30)
min3sat/c80v400l3	30	**0.96(30)**	1.84(30)	23.07(30)	46.57(17)
pmax2sat/hi	30	**0.01(29)**	9.82(29)	4.14(28)	0.15(28)
pmax2sat/me	30	**0.04(30)**	3.34(29)	0.21(28)	0.41(30)
pmax3sat/hi	30	**0.00(30)**	0.01(30)	0.04(30)	0.01(30)
Total	210	**209**	207	206	137

Table 2
Comparative results for *Dist* and other local search solvers for PMS on crafted PMS benchmark.

Instance set	#ins.	*Dist*	*CCLS*	*CCMPA*	*TWO-LEVEL*
frb	25	24.35(17)	**16.04(24)**	101.00(24)	31.79(5)
job-shop	3	**4.38(3)**	0(0)	0(0)	0(0)
maxclicque/random	96	0.28(96)	**0.00(96)**	1.36(95)	0.06(96)
maxclicque/structured	62	**11.02(60)**	13.76(49)	29.10(47)	16.20(40)
maxone/3sat	80	**1.85(80)**	4.77(78)	9.34(73)	0.00(69)
maxone/structured	60	**18.97(51)**	107.34(2)	66.40(9)	24.40(37)
min-enc/kbtree	42	**1.87(42)**	6.47(39)	119.26(21)	7.79(33)
pseudo/miplib	4	0.02(4)	**0.01(4)**	0.17(4)	0.00(3)
reversi	44	**68.98(20)**	135.76(1)	61.43(9)	1.39(4)
scheduling	5	**148.67(5)**	0(0)	0(0)	0(0)
Total	421	**378**	293	282	287

instances very quickly (one instance has no feasible solution), while the other local search solvers fail to find the best solution for some instances. Also, the averaged run time of *Dist* is usually much less than that of its competitors.

Results on Crafted PMS Benchmark: The experimental results of local search solvers on the crafted PMS benchmark are presented in Table 2. Among the 421 instances, *Dist* finds the best solution for 378 of them, while this number is 293, 282 and 287 for *CCLS*, *CCMPA* and *TWO-LEVEL* respectively. *Dist* gives the best performance for 7 out of 10 instance families, which clearly shows its superiority over its competitors on crafted PMS instances. In fact, for the other 3 families, *Dist* and the best solver for these 3 families namely *CCLS* have the same number of winning instances for 2 of the families ("maxclique/random" and "pseudo/miplib"). Overall, on the crafted PMS benchmark, *Dist* obviously has better performance than the other local search solvers, while the three competitor local search solvers (*CCLS*, *CCMPA* and *TWO-LEVEL*) have similar performance.

Results on Industrial PMS Benchmark: Table 3 summarizes the experimental results of local search solvers on the industrial PMS benchmark. *Dist* shows dramatic improvement over the other local search solvers. We observe significant performance gap between *Dist* and its competitors when comparing the solution quality they find. *Dist* finds the best solution for 384 out of 568 instances, while this figure is only 63, 112 and 38 for *CCLS*, *CCMPA* and *TWO-LEVEL* respectively. This indicates a significant improvement in the solving ability of local search solvers for industrial PMS instances.

5.3. Performance variability of local search solvers

To study the performance variability between independent runs and the robustness of the local search solvers, we conduct additional experiments to execute the local search solvers 10 times with different random seeds for some selected instances from each benchmark. For each benchmark, we select five instances from different families. The results are reported in Table 4.

Seen from the results, the performance of *Dist* is quite robust on these instances, and is better than that of other local search solvers, except for the `frb` instance where *CCLS* and *CCMPA* find better solutions. For the random instances, these local search solvers usually find the best solution steadily, and *Dist* does so with less run time. For crafted and industrial instances, the solutions found by *Dist* are always significantly better than those found by other solvers with only one exception (i.e., the `frb` instance).

5.4. Comparing Dist with complete solvers

In this subsection, we compare *Dist* and state-of-the-art complete PMS solvers. In the following, we first introduce the complete solvers in our experiments, and then present the experiment results of comparing *Dist* and complete solvers on each benchmark respectively.

Table 3
Comparative results for *Dist* and other local search solvers for PMS on industrial PMS benchmark.

Instance set	#ins.	*Dist*	*CCLS*	*CCMPA*	*TWO-LEVEL*
aes	7	129.25(2)	6.12(5)	**102.05(6)**	0.00(1)
atcoss/mesat	18	0(0)	0(0)	0(0)	0(0)
atcoss/sugar	19	**38.05(7)**	0(0)	21.22(1)	0(0)
bcp/fir	32	41.58(17)	69.02(17)	**89.90(26)**	0(0)
bcp/hipp-yRa1/simp	10	**79.02(10)**	0(0)	300.00(1)	0(0)
bcp/hipp-yRa1/su	38	**68.17(38)**	0(0)	0(0)	0(0)
bcp/msp	40	**12.67(29)**	0(0)	97.95(8)	22.04(9)
bcp/mtg	30	**38.87(27)**	0(0)	0(0)	66.28(2)
bcp/syn	38	18.89(14)	29.74(18)	**96.14(36)**	16.35(3)
circuit-trace-compaction	4	0(0)	0(0)	**53.44(1)**	0(0)
close_solutions	50	**59.97(33)**	40.80(11)	110.86(9)	34.73(6)
des	50	**105.14(13)**	0(0)	91.87(5)	0(0)
haplotype-assembly	6	**38.80(6)**	0(0)	0(0)	0(0)
hs-timetabling	2	**10.13(1)**	0(0)	0(0)	0(0)
mbd	46	**118.85(46)**	0(0)	0(0)	0(0)
packup-pms	40	**85.35(40)**	0(0)	0(0)	0(0)
pbo/mqc/nencdr	25	87.06(5)	0(0)	**57.45(11)**	0(0)
pbo/mqc/nlogencdr	25	**104.57(24)**	0(0)	0(0)	194.30(1)
pbo/routing	15	7.14(11)	121.34(1)	0(0)	**5.38(15)**
protein_ins	12	**2.19(12)**	7.34(11)	65.83(8)	0.01(1)
tpr/Multiple_path	36	**93.47(24)**	0(0)	0(0)	0(0)
tpr/One_path	25	**123.37(25)**	0(0)	0(0)	0(0)
Total	568	**384**	63	112	38

Table 4
Experimental results of *Dist*, *CCLS*, *CCMPA* and *TWO-LEVEL* with 10 runs on selected PMS instances. Each cell reports the result of a solver on an instance, where the first row reports the solution quality in the form "avg(min, max)", and the second row reports the averaged run time. If a solver fails in all runs to find a feasible solution for an instance, then the corresponding results are marked with "N/A".

Instance	*Dist* solution avg. time	*CCLS* solution avg. time	*CCMPA* solution avg. time	*TWO-LEVEL* solution avg. time
clq1-c2mv80c400l3g1.wcnf	56.0(56, 56) **0.11**	56.0(56, 56) 4.75	56.0(56, 56) 13.54	56.0(56, 56) 10.89
clq1-cv260c1040l2g1.wcnf	67.0(67, 67) **0.09**	67.0(67, 67) 0.81	67.0(67, 67) 17.08	70.7(68, 71) 174.11
file_rpms_wcnf_L2_V150_C3500_ H150_1.wcnf	679.0(679, 679) 0.00	679.0(679, 679) 0.60	679.0(679, 679) 6.65	679.0(679, 679) 0.00
file_rpms_wcnf_ L2_V150_C5000_H150_1.wcnf	1015.0(1015, 1015) 0.00	1015.0(1015, 1015) 0.00	1015.0(1015, 1015) 0.26	1015.0(1015, 1015) 0.01
file_rpms_wcnf_ L3_V100_C800_H100_1.wcnf	21.0(21, 21) 0.00	21.0(21, 21) 0.00	21.0(21, 21) 0.06	21.0(21, 21) 0.00
brock400_1.clq.wcnf	**373.6(373, 375)** 83.11	375.0(375, 375) 0.03	375.0(375, 375) 3.64	375.5(373, 376) 62.26
cnf_small.wcnf	**49.5(31, 84)** 211.25	N/A N/A	52841.8(40225, 67053) 30.36	N/A N/A
dp02s02.shuffled.cnf.wcnf	**100.0(100, 100)** 0.00	N/A N/A	1101.0(1101, 1101) 41.82	100.6(100, 101) 0.36
frb40-19-1.partial.wcnf	721.5(721, 722) 40.03	**720.0(720, 720)** 9.29	**720.0(720, 720)** 63.34	724.6(724, 725) 0.05
rev44-24.wcnf	**9.4(0, 26)** 180.50	N/A N/A	16.7(0, 66) 128.14	N/A N/A
6ebx_.1era_.g.wcnf.t.wcnf	**30.0(30, 30)** 10.41	35.4(30, 36) 5.81	34.8(30, 36) 35.12	N/A N/A
c1355_F1gat@0.wcnf	**27.5(22, 33)** 0.19	N/A N/A	133.8(84, 167) 63.36	N/A N/A
cnf.8.p.9.wcnf	**11.4(10, 13)** 137.89	N/A N/A	N/A N/A	N/A N/A
normalized-ii16a1.wcnf	**1178.9(1132, 1279)** 0.09	N/A N/A	50088.0(50088, 50088) 0.36	N/A N/A
SAT02_industrial_biere_dinphil_ dp10s10.shuffled.cnf.wcnf.8.wcnf	**326.8(240, 609)** 29.20	N/A N/A	7760.0(7760, 7760) 1.22	N/A N/A

Table 5
Comparative results of *Dist* and complete solvers on random PMS benchmark. One instance from the "pmax2sat/hi" family (file_rpms_wcnf_L2_V150_C4000_H150_1.wcnf) does not have feasible solution.

Instance set	#ins.	*Dist*	*ahmaxsat*
min2sat/v160c800l2	30	**4.20(30)**	20.20(29)
min2sat/v260c1040l2	30	**2.86(30)**	28.39(28)
min3sat/c70v350l3	30	**1.03(30)**	55.07(30)
min3sat/c80v400l3	30	**0.96(30)**	98.48(24)
pmax2sat/hi	30	0.01(29)	**4.64(30)**
pmax2sat/me	30	**0.04(30)**	1.21(30)
pmax3sat/hi	30	**0.00(30)**	29.03(30)
Total	210	**209**	201

Table 6
Comparative results of *Dist* and complete solvers on crafted PMS benchmark.

Instance set	#ins.	*Dist*	*Open-WBO-In*	*scip-maxsat*	*WPM-2014-in*
frb	25	**45.56(22)**	48.29(17)	109.52(9)	56.18(20)
job-shop	3	0(0)	**42.05(3)**	0(0)	66.31(3)
maxclicque/random	96	**0.28(96)**	33.22(58)	82.01(94)	60.07(81)
maxclicque/structured	62	**6.26(59)**	4.13(18)	54.45(31)	70.93(24)
maxone/3sat	80	**1.85(80)**	23.65(78)	5.91(80)	42.71(61)
maxone/structured	60	18.97(51)	11.49(58)	68.25(56)	**12.57(60)**
min-enc/kbtree	42	**1.87(42)**	41.80(6)	75.07(42)	128.02(5)
pseudo/miplib	4	**0.02(4)**	7.48(4)	2.86(4)	60.82(4)
reversi	44	50.11(14)	**13.41(38)**	51.33(8)	49.80(38)
scheduling	5	0(0)	0(0)	0(0)	**254.85(5)**
Total	421	**368**	280	324	301

- For the random PMS benchmark, we compare *Dist* with *ahmaxsat* [2], which won the random PMS category of the complete solvers track in the MaxSAT Evaluation 2014.
- For the crafted and industrial PMS benchmarks, we compare *Dist* with three complete solvers, namely *scip-maxsat* [48], *Open-WBO-In* [41,40] and *WPM-2014-in* [5,4]. The first two solvers are the best non-portfolio solvers in the crafted and industrial PMS category of the MaxSAT Evaluation 2014 respectively.[2] *WPM-2014-in* participated in the incomplete track and won both industrial categories of PMS and Weighted PMS, and its early versions also won several categories of the complete track in the past few evaluations [5].
- For the industrial benchmarks, we also include the complete solver *Eva500* in our experiments. As *Eva500* performs a lower bound based core-guided search, it only reports one feasible solution, when it proves optimality. Thus, it is unfair to directly compare *Eva* with other solvers which print the better-quality solution ('o' line) immediately once the solver finds one. As a reference, we report the number of solved instances for *Eva500* just to indicate the performance of the current state-of-the-art complete solver *Eva500* on industrial PMS benchmarks.

We note that, to make a fair comparison, the running time of complete solvers for solving a "winning" instance only includes the time to get the best found solution and does not include the time to prove it (if it is optimal). To this end, we executed the complete solvers with the *runsolver* software [47] to get the CPU time for each 'o' line, and the time for the last 'o' line was recorded.

Results on Random PMS Benchmark: The results comparing *Dist* and *ahmaxsat* on random PMS instances are presented in Table 5, which clearly show that *Dist* performs better than the complete solver *ahmaxsat* on random PMS instances, in terms of the solution quality and the run time. For the instances where *ahmaxsat* finds the optimal solution, *Dist* finds the optimal solutions with much less averaged time. For the other instances (except one instance which has no feasible solution), *Dist* finds better solutions than *ahmaxsat*, and we tend to believe those solutions are optimal. Overall, the number of "winning" instance of *Dist* is 8 more than that of *ahmaxsat*.

Results on Crafted PMS Benchmark: The experimental results comparing *Dist* and complete solvers on the crafted PMS benchmark are presented in Table 6. Among 421 instances, *Dist* finds the best solution for 368 of them, more than all the complete solvers. In detail, *Dist* gives the best performance for 6 out of 10 instance families. For the remaining 4 families, 2 of them are dominated by *Open-WBO-In* while the other 2 families are dominated by *WPM-2014-in*. We also note that *Dist* is the only local search solver that performs better than all complete solvers on crafted PMS instances, as witnessed by the results of the MaxSAT Evaluation 2014.

Results on Industrial PMS Benchmark: Table 7 summarizes the experimental results comparing *Dist* and complete solvers on the industrial PMS benchmark. Overall, for these industrial instances, *Dist* achieves similar performance with the ILP-

[2] http://www.maxsat.udl.cat/14/results/index.html.

Table 7
Comparative results for *Dist* and complete solvers on industrial PMS benchmark.

Instance set	#ins.	#prov. by *Eva*	*Dist*	*Open-WBO-In*	*scip-maxsat*	*WPM-2014-in*
aes	7	1	178.31(3)	24.43(1)	**131.44(5)**	0(0)
atcoss/mesat	18	8	0(0)	150.12(9)	0(0)	**160.92(10)**
atcoss/sugar	19	11	0(0)	**43.16(15)**	0(0)	118.80(13)
bcp/fir	32	29	46.36(15)	**7.50(32)**	45.29(28)	15.96(29)
bcp/hipp-yRa1/simp	10	8	98.10(7)	30.95(9)	0(0)	**46.29(10)**
bcp/hipp-yRa1/su	38	31	79.92(28)	**6.00(31)**	0(0)	43.01(30)
bcp/msp	40	16	**13.12(28)**	98.86(9)	87.32(15)	90.36(13)
bcp/mtg	30	30	32.56(12)	**0.09(30)**	126.45(14)	0.16(30)
bcp/syn	38	17	25.36(14)	32.03(13)	**20.56(35)**	49.11(6)
circuit-trace-compaction	4	3	0(0)	**27.47(4)**	114.91(1)	46.29(4)
close_solutions	50	29	56.78(27)	**32.47(46)**	67.27(6)	64.02(42)
des	50	29	0(0)	51.78(30)	0(0)	**181.24(42)**
haplotype-assembly	6	5	0(0)	1.35(5)	195.07(1)	**0.32(5)**
hs-timetabling	2	1	**10.13(1)**	56.44(1)	0(0)	18.21(1)
mbd	46	42	0(0)	**12.33(46)**	281.01(2)	50.54(34)
packup-pms	40	40	47.25(6)	38.62(40)	15.00(38)	**6.26(40)**
pbo/mqc/nencdr	25	25	0(0)	**61.57(21)**	0(0)	65.63(21)
pbo/mqc/nlogencdr	25	25	289.02(1)	35.18(25)	286.42(1)	**11.69(25)**
pbo/routing	15	15	7.14(11)	**0.36(15)**	86.17(11)	0.77(15)
protein_ins	12	4	**2.19(12)**	111.73(10)	8.05(1)	164.11(5)
tpr/Multiple_path	36	30	17.50(2)	**73.06(28)**	0(0)	129.46(22)
tpr/One_path	25	25	0(0)	**38.27(24)**	278.13(3)	140.41(24)
Total	568	424	167	**444**	161	421

Table 8
Experimental results comparing *Dist* with the portfolios solver *ISAC+−pms 2014*. The 'VBS' column reports the results for Virtual Best Solver, which hypothetically calls *Dist* for the instances where *Dist* has better performance and calls *ISAC+−pms* for the instances where *ISAC+−pms* has better performance.

Benchmark	#ins.	*Dist*	*ISAC+−pms*	VBS
Random PMS	210	**1.31(209)**	50.23(166)	1.34(210)
Crafted PMS	421	8.37(366)	**25.71(403)**	13.92(421)
Industrial PMS	568	43.46(161)	**53.57(540)**	50.55(556)

based solver *scip-maxsat*, but still cannot compete with the core-based complete solvers. Nevertheless, *Dist* gives the best performance for 3 instance families, namely "bcp/hipp-yRa1/simp", "bcp/msp" and "protein_ins".

To sum up, when compared with complete solvers, *Dist* performs better on random and crafted PMS instances. But the performance of *Dist* is worse than core-based complete solvers on industrial instances, although it represents a significant improvement over previous local search solvers.

5.5. Comparing Dist with portfolio solver

In this subsection, we compare *Dist* with a state-of-the-art portfolio solver for PMS, namely *ISAC+−pms* [6], which showed better performance than all non-portfolio solvers and won the crafted and industrial categories of the complete solvers track in the MaxSAT Evaluation 2014.

As with complete solvers, the running time of *ISAC+−pms* for solving a "winning" instance only includes the time to get the best found solution and does not include the time to prove it (if it is optimal). To this end, we executed *ISAC+−pms* with the *runsolver* software. Originally, *ISAC+−pms* outputs all 'o' lines just before its termination. For our experiments, we modified the `python` script of *ISAC+−pms* to let *ISAC+−pms* output the 'o' line in a successive way so that we could record the correct run time for getting the best found solution (the last 'o' line). We also report the results for a Virtual Best Solver (that is, for each instance it hypothetically calls the best solver for the instance to solve it), to see how the performance of *Dist* can contribute to an expanded portfolio, under the slightly idealizing assumption that *Dist* could be called for all instances on which it performs better than the existing portfolio.

The results comparing *Dist* with the portfolio solver *ISAC+−pms* are presented in Table 8. It is clear that *Dist* has better performance than *ISAC+−pms* on the random PMS benchmark, while *ISAC+−pms* has stronger performance on the crafted and industrial benchmarks. The results of Virtual Best Solver are better than *ISAC+−pms* on all the benchmarks. This indicates that including *Dist* into the portfolio solver would make a stronger portfolio solver on all the benchmarks.

5.6. Effectiveness of the underlying ideas in Dist

In this section, we conduct further empirical analyses to study effectiveness of the key components in *Dist*, including the clause weighting scheme that works only for hard clauses, as well as strategies in the variable selection. We compare *Dist* with its four alternatives, which are modified from *Dist* as follows:

Table 9
Experimental results comparing *Dist* with its variants. For each benchmark, each cell reports for each solver the number of "winning" instances in the benchmark and the averaged run time over these "winning" instances.

Benchmark	#ins.	*Dist*	*Dist*_alt1	*Dist*_alt2	*Dist*_alt3	*Dist*_alt4
Random PMS	210	**1.31(209)**	32.06(49)	0.01(60)	20.41(163)	3.57 (207)
Crafted PMS	421	**9.39(375)**	15.11(341)	10.32(352)	13.53(353)	10.84(330)
Industrial PMS	568	**68.81(277)**	70.58(200)	56.52(182)	74.58(213)	54.01(149)

- *Dist*_alt1: update clause weights for all clauses (replace line 12 in Algorithm 1). This variant is tested in order to study the effectiveness of clause weighting scheme that works only for hard clauses.
- *Dist*_alt2: when hard-decreasing variables exist, pick the one with the best *hscore*, breaking ties in favor of the one with the greatest *sscore* (replace lines 7–8 in Algorithm 1).
- *Dist*_alt3: when hard-decreasing variables exist, pick one of them uniformly at random. (replace lines 7–8 in Algorithm 1). Both *Dist*_alt2 and *Dist*_alt3 are tested in order to study the effectiveness of the BMS heuristic in choosing a hard-decreasing variable to be flipped.
- *Dist*_alt4: always pick a random variable from the selected falsified clause (replace lines 15–16 in Algorithm 1). This variant is tested in order to study the effectiveness of the idea of combining random walk with a greedy strategy that chooses a variable with the greatest soft score from the selected falsified clause.

The experimental results (Table 9) demonstrate the superiority of *Dist* over these variants, which indicates the effectiveness of the algorithmic components. Specifically, the comparison between *Dist* and *Dist*_alt1 indicates the effectiveness of the clause weighting scheme that works only for hard clauses; the superiority of *Dist* over *Dist*_alt2 and *Dist*_alt3 demonstrate the effectiveness of the BMS (Best from Multiple Selections) heuristic in choosing the hard-decreasing variable for flipping; finally, the comparison between *Dist* and *Dist*_alt4 indicates the effectiveness of the idea of combining random walk with a greedy strategy for choosing the flipping variable from the selected falsified clause.

We also note that these degraded alternatives of *Dist* still exhibit significantly better performance than previous local search solvers on crafted and industrial benchmarks. This indicates that heuristics based on separated hard score and soft score are more promising for solving structured PMS instances than previous local search methods.

6. Improving *Dist* on industrial PMS instances

The preceding section has shown the excellent performance of *Dist* on random and crafted PMS instances, which is better than that of state-of-the-art solvers, including both complete and incomplete (i.e. local search) solvers. However, for industrial PMS instances, although *Dist* significantly outperforms other local search solvers, its performance is still unsatisfactory, and there is an obvious gap between the performance of *Dist* and state-of-the-art complete solvers.

In this section, we aim to improve *Dist* on industrial PMS instances. To this end, we propose a procedure for generating a good initial assignment for local search algorithms for PMS and utilize it to improve *Dist*, resulting in a new local search PMS solver called *DistUP*.

6.1. Initialization via priority unit propagation

In this subsection, we propose a procedure of generating an initial assignment for local search algorithms for PMS. The procedure employs an unit propagation (UP) technique that puts priority on hard unit clauses, and thus is named *PrioUP*.

Before we present the details of the *PrioUP* procedure, we first introduce the key data structures. The array *value* records the assigned value for each variable. For each variable x, $value(x)$ has 4 possible values $\{-2, -1, 0, 1\}$, and their meanings are explained as below:

- $value(x) = -2$ means x is forbidden to be assigned. When unit clauses l and $\neg l$ appear in F simultaneously (due to previous propagations), the corresponding variable x is forbidden to be assigned, as assigning x would cause a conflict.
- $value(x) = -1$ means x is unassigned.
- $value(x) = 0$ means x is assigned with the value 0.
- $value(x) = 1$ means x is assigned with the value 1.

The queue Q is a priority queue that stores all unit clauses for propagation. In Q, hard unit clauses have higher priority than soft ones. All hard unit clauses have the same priority and are managed in an FIFO (First In First Out) manner, and so do the soft ones. Since each unit clause has only one literal, Q indeed stores those literals. Note that we avoid duplication (i.e., a literal appears more than one time) in Q, and also avoid including opposite literals in Q. This is accomplished via an auxiliary array $polarity\_in\_Q$.

The auxiliary array $polarity\_in\_Q$ records the information for each variable with regards to its appearance in Q. For each variable x, $polarity\_in\_Q(x)$ records whether x appears in Q, and the polarity of the literal when x appears in Q. It has 3 possible values $\{-1, 0, 1\}$, as explained below:

- $polarity\_in\_Q(x) = -1$ means x does not appear in Q (we say a variable x appears in Q, if Q contains either x or $\neg x$).
- $polarity\_in\_Q(x) = 0$ means the literal $\neg x$ is in Q.
- $polarity\_in\_Q(x) = 1$ means the literal x is in Q.

The *PrioUP* procedure is outlined in Algorithm 2,

Algorithm 2: PrioUP (F).

```
Input: A CNF formula F
Output: An assignment of variables in F
1  For each variable x, value(x) := −1, polarity_in_Q(x) := −1;
2  Build the priority queue Q by inserting all unit clauses into Q;
3  while Q ≠ ∅ do
4      l :=GetFirstLiteral(Q) and x := var(l);
5      remove l out of Q;
6      if value(x) = −1 then
7          value(x) := polarity(l);
8          F := F|_l;
9          foreach newly generated unit clause p do
10             y := var(p);
11             if polarity_in_Q(y) = −1 then
12                 enqueue p in Q;
13                 polarity_in_Q(y) := polarity(p);
14             else if polarity_in_Q(y) ≠ polarity(p) then
15                 value(y) := −2;
16 foreach x with value(x) = −1 or value(x) = −2 do
17     value(x) := a random number from {0, 1} ;
18 return value;
```

and explained as follows. In the beginning, each variable is marked as "unassigned", and all unit clauses in the formula F are added into the queue Q.

While Q is not empty, the first literal (or say, unit clause) l in Q is extracted from Q and propagated. The variable corresponding to literal l is denoted as x. We know that x could not have been propagated (i.e., $value(x)$ could not be 0 or 1), as otherwise the formula as well as Q would not contain any literal of x. Thus, $value(x)$ is either -1 or -2.

If $value(x) = -1$, which means x is unassigned, then x is assigned to the value of $polarity(l)$ to satisfy the unit clause l, and F is conditioned on the literal l. If this process generates any new unit clause, for each such unit clause p, we do the following checking and operations if needed. Let y denote the corresponding variable of p. If neither literal p nor literal $\neg p$ is in Q (i.e., $polarity\_in\_Q(y) = -1$), then p is enqueued into Q, and $polarity\_in\_Q(y)$ is set to $polarity(p)$. Otherwise, if literal p is already in Q, we do nothing (we do not add p into Q for avoiding duplication). If literal $\neg p$ is in Q (i.e., $polarity\_in\_Q(y) \neq polarity(p)$), then variable y is marked as "forbidden" (by setting $value(y)$ to -2), as p and $\neg p$ appear in the formula simultaneously and assigning y would cause a conflict.

If $value(x) = -2$, which means x is forbidden to be propagated, we just ignore it. A principle in our unit propagation initialization procedure is to avoid generating conflicts. Thus, for variables marked as "forbidden", we would rather leave them unassigned during the unit propagation procedure, as assigning them would cause empty clauses.

Finally, after Q becomes empty, if there are variables unassigned, including those with $value(x) = -1$ and also those with $value(x) = -2$, then these variables are assigned 0 or 1 uniformly randomly.

6.2. The DistUP solver

We apply the *PrioUP* procedure to improve the *Dist* solver, leading to the *DistUP* solver. This is done by modifying only a few lines of codes of *Dist*.

In the following, we illustrate how to apply *PrioUP* to improve a local search PMS solver. To facilitate our discussions, we assume that local search algorithms allow to accept a specific assignment as its initialized assignment.[3] Let us recall the general scheme of local search algorithms for PMS. Starting with a complete assignment α, a local search algorithm chooses a variable and flips it in each search step. Whenever finding a feasible assignment with a lower cost, the algorithm records it as the best found assignment. To apply the *PrioUP* procedure to a local search solver, we first generate a complete assignment α using *PrioUP*, and then execute the local search solver on the original PMS instance with α as its initial assignment. Note that the instance is modified in the *PrioUP* procedure, but we use the original instance for local search.

[3] For this, we only need to perform minor modifications to the initialization of the given local search procedure.

Table 10
Comparative results for *DistUP* and *Dist* on the industrial PMS benchmark from the MaxSAT Evaluation 2014.

Instance set	#ins.	*Dist*	*DistUP*
aes	7	**134.21(6)**	166.48(4)
atcoss/mesat	18	0(0)	0(0)
atcoss/sugar	19	131.94(1)	**148.08(6)**
bcp/fir	32	56.05(26)	**35.43(30)**
bcp/hipp-yRa1/simp	10	**79.02(10)**	20.08(7)
bcp/hipp-yRa1/su	38	**68.35(35)**	45.83(32)
bcp/msp	40	0.28(22)	**42.95(30)**
bcp/mtg	30	51.35(13)	**42.66(27)**
bcp/syn	38	63.76(30)	**74.19(34)**
circuit-trace-compaction	4	0(0)	0(0)
close_solutions	50	52.90(29)	**28.64(42)**
des	50	89.04(9)	**153.50(16)**
haplotype-assembly	6	1.26(1)	**75.04(5)**
hs-timetabling	2	0(0)	**232.53(1)**
mbd	46	121.29(24)	**21.90(25)**
packup-pms	40	57.34(9)	**65.56(38)**
pbo/mqc/nencdr	25	89.97(4)	**121.38(8)**
pbo/mqc/nlogencdr	25	**133.61(14)**	100.62(11)
pbo/routing	15	7.14(11)	**25.46(15)**
protein_ins	12	**2.19(12)**	3.40(12)
tpr/Multiple_path	36	**78.62(15)**	71.14(9)
tpr/One_path	25	**161.41(16)**	87.71(10)
Total	568	287	**362**

That is, we do not fix the value of any variable and all variables are treated equally during the local search. In this way, variables which are assigned incorrectly by *PrioUP* have a chance to be corrected.

6.3. Empirical evaluations of DistUP

In this subsection, we evaluate the performance of *DistUP* on industrial PMS instances from the MaxSAT Evaluation 2014. We observe that almost all these industrial PMS instances have unit clauses. Indeed, as stated in [39], most industrial MaxSAT and SAT instances have a non-negligible number of unit clauses, and researchers have pointed out the importance of such clauses. The *PrioUP* procedure can be considered as a step towards exploiting unit clauses in local search solvers for PMS.

We first compare *DistUP* with the *Dist* solver from Section 4, and then compare *DistUP* with state-of-the-art complete solvers on industrial PMS instances. Additionally, we also compare *DistUP* with *Dist* and a state-of-the-art complete solver on industrial Weighted PMS instances from the MaxSAT Evaluation 2014. The experimental environment and protocol in this section are the same as those used in Section 5. Also note that, the run time for *DistUP* always includes the time for *PrioUP*. Indeed, *PrioUP* terminates in one second for all the instances in our experiments.

6.3.1. Experiments of DistUP on industrial PMS instances

The experimental results comparing *Dist* and *DistUP* on the industrial PMS benchmark are presented in Table 10, which clearly demonstrate the significant improvement of *DistUP* over *Dist*. Among the 568 industrial PMS instances, *DistUP* have 362 "winning" instances, compared to 287 for *Dist*. *DistUP* has better performance than *Dist* on 13 instance families, and has worse performance on 7 families. According to these experimental results, the performance of local search solvers on industrial PMS instances can be improved by using the *PrioUP* procedure to generate the initial assignment.

We also compare *DistUP* with complete solvers on industrial PMS instances, and the experiment results are presented in Table 11. Although *DistUP* usually finds better solutions than *Dist* on industrial PMS instances, the solution quality returned by *DistUP* is still worse than complete solvers on many instances, and thus *DistUP* does not show an obvious increase in the number of "winning" instances. Nevertheless, *DistUP* does have 24 more "winning" instances than *Dist* when compared to the same complete solvers. Indeed, like the case for SAT, improving local search solvers on industrial benchmarks of PMS (or other MaxSAT variants) remains a big challenge. This has been emphasized in a recent paper [24]. Although *DistUP* cannot yet rival complete solvers on industrial benchmarks, experiments comparing *DistUP* with *Dist* and previous local search solvers show significant progress in this direction.

6.3.2. Experiments of DistUP on industrial weighted PMS instances

In order to study the effectiveness of the *PrioUP* procedure on Weighted PMS industrial instances, we also compare *Dist* and *DistUP* on Weighted PMS industrial instances from the MaxSAT Evaluation 2014. We utilize the automatic configuration tool *SMAC* [30] to tune the parameter settings for *Dist* on the Weighted PMS industrial benchmark from the MaxSAT Evaluation 2014, and the suggested setting is: $t = 18$, $wp = 0.1$ and $sp = 0.013$. We use this parameter setting for both *Dist* and *DistUP*.

Table 11
Comparative results for *DistUP* and complete solvers on the industrial PMS benchmark.

Instance set	#ins.	#prov. by *Eva*	*DistUP*	*Open-WBO-In*	*scip-maxsat*	*WPM-2014-in*
aes	7	1	180.83(3)	24.43(1)	**131.44(5)**	0(0)
atcoss/mesat	18	8	0(0)	150.12(9)	0(0)	**160.91(10)**
atcoss/sugar	19	11	71.38(1)	**45.82(14)**	0(0)	118.80(13)
bcp/fir	32	29	26.45(14)	**7.50(32)**	45.29(28)	15.96(29)
bcp/hipp-yRa1/simp	10	8	21.21(5)	30.95(9)	0(0)	**46.29(10)**
bcp/hipp-yRa1/su	38	31	45.98(27)	**6.00(31)**	0(0)	49.86(31)
bcp/msp	40	16	**42.95(30)**	98.86(9)	87.32(15)	90.36(13)
bcp/mtg	30	30	32.46(19)	**0.09(30)**	126.45(14)	0.16(30)
bcp/syn	38	17	35.29(15)	32.03(13)	**20.56(35)**	49.11(6)
circuit-trace-compaction	4	3	0(0)	**27.47(4)**	114.91(1)	46.29(4)
close_solutions	50	29	30.96(38)	**32.47(46)**	67.27(6)	64.02(42)
des	50	29	15.39(1)	51.79(30)	0(0)	**178.69(41)**
haplotype-assembly	6	5	0(0)	1.35(5)	195.07(1)	**0.32(5)**
hs-timetabling	2	1	232.53(1)	56.44(1)	0(0)	**18.21(1)**
mbd	46	42	0(0)	**12.33(46)**	281.01(2)	50.54(34)
packup-pms	40	40	0.82(6)	38.62(40)	15.00(38)	**6.26(40)**
pbo/mqc/nencdr	25	25	0(0)	**61.57(21)**	0(0)	65.63(21)
pbo/mqc/nlogencdr	25	25	45.02(2)	35.18(25)	286.42(1)	**11.69(25)**
pbo/routing	15	15	25.46(15)	**0.36(15)**	86.17(11)	0.77(15)
protein_ins	12	4	**3.40(12)**	111.73(10)	8.05(1)	164.11(5)
tpr/Multiple_path	36	30	146.72(2)	**73.06(28)**	0(0)	129.46(22)
tpr/One_path	25	25	0(0)	**38.27(24)**	278.13(3)	140.41(24)
Total	568	424	191	**443**	161	421

Table 12
Comparative results for Dist and DistUP on the industrial Weighted PMS benchmark.

Instance class	#ins.	*Dist*	*DistUP*
dustrial/haplotyping-pedigrees	100	62.19(35)	**123.76(93)**
hs-timetabling	14	**190.12(3)**	0.00(0)
packup-wpms	99	115.38(20)	**81.04(79)**
industrial/preference_planning	29	0.00(1)	**36.99(25)**
timetabling	26	**140.47(7)**	115.07(1)
ustrial/upgradeability-problem	100	0.23(3)	**0.09(97)**
wcsp/spot5/dir	21	**65.81(14)**	45.67(12)
wcsp/spot5/log	21	94.29(16)	**61.39(18)**
Total	410	99	**325**

Table 13
Comparative results for DistUP and WPM-2014-in on the industrial Weighted PMS benchmark.

Instance class	#ins.	#prov. by *Eva*	*DistUP*	*WPM-2014-in*
dustrial/haplotyping-pedigrees	100	97	12.13(30)	**29.55(100)**
hs-timetabling	14	0	0.00(0)	**241.10(14)**
packup-wpms/	99	99	0.03(2)	**17.81(99)**
industrial/preference_planning	29	28	13.93(7)	**16.41(29)**
timetabling	26	10	38.74(2)	**134.02(21)**
ustrial/upgradeability-problem	100	100	0.00(0)	**1.07(100)**
wcsp/spot5/dir	21	14	13.16(6)	**73.79(21)**
wcsp/spot5/log	21	14	**78.38(17)**	50.51(9)
Total	410	362	64	**393**

The experimental results are summarized in Table 12, which show that *DistUP* obtains better solutions than *Dist* on most of the instances. Overall, *DistUP* finds a better or equal quality solution on 325 instances among the 410 instances. Specifically, *DistUP* performs better on 5 instance families, and worse on the other 3 families. These experimental results indicate that the *PrioUP* procedure is also beneficial for solving Weighted PMS in most cases.

However, the performance of *DistUP* on industrial instances of Weighted PMS is still much worse than complete solvers. For example, we compared *DistUP* with *WPM-2014-in* on these Weighted PMS industrial instances, and the results (Table 13) show that the number of "winning" instances of *WPM-2014-in* is much more than that of *DistUP*. Nevertheless, we would like to note that *DistUP* performs much better than *WPM-2014-in* on one family namely "wcsp/spot5/log". In this sense, it could be complementary to some extent to complete solvers for solving Weighted PMS industrial instances.

6.4. Related work and discussions on PrioUP

There is some work on improving local search algorithms by constructing effective initial solutions, most of which are studied in the context of the Most Probable Explanation (MPE) in Bayesian networks [32,28,42]. With regard to MaxSAT, Hains et al. transformed MaxSAT to Walsh polynomials and compute the hyperplane averages, which are then used to generate the initial assignment [21]. Their experiments on unweighted industrial MaxSAT benchmark from the MaxSAT Evaluation 2012 show that, by replacing the randomized initialization with the hyperplane-based initialization, better solutions can be obtained.

Unit propagation preserves the satisfiability of the instance but does not preserve the number of falsified clauses for every assignment. Due to this fact, the use of UP was mainly restricted to the SAT problem, and UP was not applied to local search algorithms for MaxSAT until recently. To the best of our knowledge, there are only two studies using UP in local search algorithms for MaxSAT in the literature [22,1], and we are not aware of any work using UP in local search algorithms for PMS. Heras and Bañeres [22] proposed several preprocessors for MaxSAT and applied them before local search algorithms. One of the preprocessors is the UP preprocessor, which iteratively performs the following operations until no more non-propagated unit clause remains: it first executes a simulation of unit propagation (i.e., the consequences are not applied to the formula) to find a conflicting clause, and then builds the corresponding refutation tree,[4] and finally it applies MaxSAT resolution as indicated by the refutation tree to transform the formula. Abramé and Habet [1] integrated the UP technique into a local search algorithm, by applying UP after each flip step to dynamically build refutation trees and detect conflicts. The processing technique in [1] is similar to the one in [22], but it is used dynamically during the search process. Both of these two methods are mainly evaluated on random and crafted unweighted MaxSAT instances.

7. Conclusions and future work

In this work, we proposed a number of novel ideas for local search for Partial MaxSAT, which exploit the distinction between hard and soft clauses. Specifically, we proposed a clause weighting scheme that works only for hard clauses, the idea of separating hard and soft score, and a variable selection heuristic based on hard score and soft score. We then used these ideas to develop a local search algorithm for PMS called *Dist*. Experimental results show that *Dist* dramatically outperforms previous local search algorithms. Also, *Dist* outperforms complete algorithms on random and crafted benchmarks, but is still worse on industrial instances. Further, we proposed an initialization procedure that makes use of unit propagation and puts priority on hard unit clauses, and applied it to improve *Dist* on industrial instances, resulting in the *DistUP* solver. Experimental results show that *DistUP* significantly improves *Dist* on industrial PMS and WPMS instances, yet it cannot rival state-of-the-art complete solvers.

This work made a breakthrough in local search for PMS, which was also confirmed by the excellent performance of *Dist* in the MaxSAT Evaluation 2014. The strong experimental results suggested that local search based on hard and soft score is a promising direction for solving PMS and deserves further research, and we would like to extend these methods to weighted Partial MaxSAT. Another interesting direction is to study the initialization methods for MaxSAT problems.

Acknowledgements

This work is supported in part by the China National 973 Program under Project 2014CB340301 and National Natural Science Foundation of China 61502464, 61370072 and 61572234. Chuan Luo is also supported by the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (Grant 2016A06), and Kaile is also supported by ARC DP150101618. We would like to thank the anonymous reviewers for their helpful comments on the earlier versions of this paper.

References

[1] André Abramé, Djamal Habet, Inference rules in local search for Max-SAT, in: IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI, Athens, Greece, 2012, pp. 207–214.
[2] André Abramé, Djamal Habet, Ahmaxsat: description and evaluation of a branch and bound Max-SAT solver, J. Satisf. Boolean Model. Comput. 9 (2015) 89–128.
[3] Carlos Ansótegui, Joel Gabàs, Solving (weighted) partial MaxSAT with ILP, in: Proceedings of the 10th International Conference of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR, Yorktown Heights, NY, USA, 2013, pp. 403–409.
[4] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, Jordi Levy, Improving WPM2 for (weighted) partial MaxSAT, in: Proceedings of the 19th International Conference of Principles and Practice of Constraint Programming, CP, Uppsala, Sweden, 2013, pp. 117–132.
[5] Carlos Ansótegui, Maria Luisa Bonet, Jordi Levy, SAT-based MaxSAT algorithms, Artif. Intell. 196 (2013) 77–105.
[6] Carlos Ansótegui, Yuri Malitsky, Meinolf Sellmann, MaxSAT by improved instance-specific algorithm configuration, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI, Québec City, Québec, Canada, 2014, pp. 2594–2600.
[7] Josep Argelich, Chu Min Li, Felip Manyà, Jordi Planes, The MaxSAT evaluations 2010–2015, http://www.maxsat.udl.cat.

[4] A refutation tree for an unsatisfiable clause set is a resolution process that derives an empty clause, where every clause is used exactly once during the resolution process.

[8] Adrian Balint, Andreas Fröhlich, Improving stochastic local search for SAT with a new probability distribution, in: Proceedings of the 13th International Conference of Theory and Applications of Satisfiability Testing, SAT, Edinburgh, UK, 2010, pp. 10–15.
[9] Daniel Le Berre, Anne Parrain, The Sat4j library, release 2.2, J. Satisf. Boolean Model. Comput. 7 (2–3) (2010) 59–64.
[10] Shaowei Cai, Kaile Su, Local search for Boolean Satisfiability with configuration checking and subscore, Artif. Intell. 204 (2013) 75–98.
[11] Shaowei Cai, Kaile Su, Abdul Sattar, Local search with edge weighting and configuration checking heuristics for minimum vertex cover, Artif. Intell. 175 (9–10) (2011) 1672–1696.
[12] Shaowei Cai, Chuan Luo, John Thornton, Kaile Su, Tailoring local search for partial MaxSAT, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI, Québec City, Québec, Canada, 2014, pp. 2623–2629.
[13] Shaowei Cai, Chuan Luo, Kaile Su, CCAnr: a configuration checking based local search solver for non-random satisfiability, in: Proceedings of 18th International Conference of Theory and Applications of Satisfiability Testing, SAT, Austin, TX, USA, 2015, pp. 1–8.
[14] 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, 2015, pp. 747–753.
[15] Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, Shuichi Miyazaki, Local search algorithms for partial MaxSAT, in: Proceedings of the 14th National Conference on Artificial Intelligence, AAAI, Providence, Rhode Island, 1997, pp. 263–268.
[16] Jessica Davies, 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, 2011, pp. 225–239.
[17] Jessica Davies, Jeremy Cho, Fahiem Bacchus, Using learnt clauses in MaxSAT, in: Proceedings of the 16th International Conference Principles and Practice of Constraint Programming, CP, St. Andrews, Scotland, UK, 2010, pp. 176–190.
[18] Jessica Davies, Solving MaxSAT by decoupling optimization and satisfaction, PhD thesis, 2013.
[19] Zhaohui Fu, 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, 2006, pp. 252–265.
[20] Ana Graça, João Marques-Silva, Inês Lynce, Arlindo L. Oliveira, Haplotype inference with pseudo-Boolean optimization, Ann. Oper. Res. 184 (1) (2011) 137–162.
[21] Doug Hains, Darrell Whitley, Adele E. Howe, Wenxiang Chen, Hyperplane initialized local search for MAXSAT, in: Proceedings of Genetic and Evolutionary Computation Conference, GECCO, Amsterdam, The Netherlands, 2013, pp. 805–812.
[22] Federico Heras, David Bañeres, The impact of Max-SAT resolution-based preprocessors on local search solvers, J. Satisf. Boolean Model. Comput. 7 (2–3) (2010) 89–126.
[23] Federico Heras, Javier Larrosa, Albert Oliveras, MiniMaxSAT: an efficient weighted Max-SAT solver, J. Artif. Intell. Res. 31 (2008) 1–32.
[24] Marijn J.H. Heule, Torsten Schaub, What's hot in the SAT and ASP competitions, in: Proceedings of the 29th AAAI Conference on Artificial Intelligence, AAAI, Austin, Texas, USA, 2015, pp. 4322–4323.
[25] Holger H. Hoos, Thomas Stützle, Stochastic Local Search: Foundations & Applications, Elsevier/Morgan Kaufmann, 2005.
[26] Holger H. Hoos, On the run-time behaviour of stochastic local search algorithms for SAT, in: Proceedings of the 16th National Conference on Artificial Intelligence, AAAI, Orlando, Florida, USA, 1999, pp. 661–666.
[27] Frank Hutter, Dave A.D. Tompkins, Holger H. Hoos, Scaling and probabilistic smoothing: efficient dynamic local search for SAT, in: Proceedings of the 8th International Conference of Principles and Practice of Constraint Programming, CP, Ithaca, NY, USA, Springer, 2002, pp. 233–248.
[28] Frank Hutter, Holger H. Hoos, Thomas Stützle, Efficient stochastic local search for MPE solving, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI, Edinburgh, Scotland, UK, 2005, pp. 169–174.
[29] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Thomas Stützle, ParamILS: an automatic algorithm configuration framework, J. Artif. Intell. Res. 36 (2009) 267–306.
[30] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proceedings of the 5th International Conference of Learning and Intelligent Optimization, LION, Rome, Italy, 2011, pp. 507–523.
[31] Yuejun Jiang, Henry Kautz, Bart Selman, Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT, in: First International Joint Workshop on Artificial Intelligence and Operations Research, 1995.
[32] K. Kask, R. Dechter, Stochastic local search for Bayesian networks, in: Proceedings of the 7th International Workshop on Artificial Intelligence and Statistics, AISTATS, Fort Lauderdale, Florida, US, 1999.
[33] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, Ryuzo Hasegawa, QMaxSat: a partial Max-SAT solver, J. Satisf. Boolean Model. Comput. 8 (1/2) (2012) 95–100.
[34] Chu Min Li, Yu Li, Satisfying versus falsifying in local search for satisfiability – (poster presentation), in: Alessandro Cimatti, Roberto Sebastiani (Eds.), Proceedings of the 15th International Conference of Theory and Applications of Satisfiability Testing, SAT, Trento, Italy, 2012, pp. 477–478.
[35] Chu Min Li, Felip Manyà, Jordi Planes, New inference rules for Max-SAT, J. Artif. Intell. Res. 30 (2007) 321–359.
[36] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, Jordi Planes, Exploiting cycle structures in Max-SAT, in: Proceedings of the 12th International Conference of Theory and Applications of Satisfiability Testing, SAT, Swansea, UK, 2009, pp. 467–480.
[37] Han Lin, Kaile Su, Chu Min Li, Within-problem learning for efficient lower bound computation in Max-SAT solving, in: Proceedings of the 23rd AAAI Conference on Artificial Intelligence, AAAI, Chicago, Illinois, USA, 2008, pp. 351–356.
[38] Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, Kaile Su, CCLS: an efficient local search algorithm for weighted maximum satisfiability, IEEE Trans. Comput. 64 (7) (2015) 1830–1843.
[39] Zoltán Ádám Mann, Typical-case complexity and the SAT competitions, in: Proceedings of the 5th Pragmatics of SAT Workshop, POS@SAT, Vienna, Austria, 2014, pp. 72–87.
[40] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, Inês Lynce, Incremental cardinality constraints for MaxSAT, in: Proceedings of the 20th International Conference of Principles and Practice of Constraint Programming, CP, Lyon, France, 2014, pp. 531–548.
[41] Ruben Martins, Vasco M. Manquinho, Inês Lynce, Open-WBO: a modular MaxSAT solver, in: Proceedings of the 17th International Conference of Theory and Applications of Satisfiability Testing, Vienna, Austria, 2014, pp. 438–445.
[42] Ole J. Mengshoel, David C. Wilkins, Dan Roth, Initialization and restart in stochastic local search: computing a most probable explanation in Bayesian networks, IEEE Trans. Knowl. Data Eng. 23 (2) (2011) 235–247.
[43] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, Joao Marques-Silva, Iterative and core-guided MaxSAT solving: a survey and assessment, Constraints 18 (4) (2013) 478–534.
[44] António Morgado, Carmine Dodaro, Joao Marques-Silva, Core-guided maxsat with soft cardinality constraints, in: Proceedings of the 20th International Conference of Principles and Practice of Constraint Programming, CP, Lyon, France, 2014, pp. 564–573.
[45] Nina Narodytska, Fahiem Bacchus, Maximum satisfiability using core-guided MaxSAT resolution, in: Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI, Québec City, Québec, Canada, 2014, pp. 2717–2723.
[46] Duc Nghia Pham, John Thornton, Charles Gretton, Abdul Sattar, Advances in local search for satisfiability, in: Proceedings of the 20th Australian Joint Conference on Artificial Intelligence, AI, Gold Coast, Australia, Springer, 2007, pp. 213–222.
[47] Olivier Roussel, Controlling a solver execution with the runsolver tool, J. Satisf. Boolean Model. Comput. 7 (4) (2011) 139–144.
[48] Masahiro Sakai, scip-maxsat, https://github.com/msakai/scip-maxsat.

[49] Kevin Smyth, Holger H. Hoos, Thomas Stützle, Iterated robust tabu search for MAX-SAT, in: Proceedings of the 16th Canadian Conference on AI, Halifax, Canada, 2003, pp. 129–144.
[50] John Thornton, Abdul Sattar, Dynamic constraint weighting for over-constrained problems, in: Proceedings of the 5th Pacific Rim International Conference on Artificial Intelligence, PRICAI, Singapore, 1998, pp. 377–388.
[51] John Thornton, Stuart Bain, Abdul Sattar, 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, 2002, pp. 603–614.
[52] John Thornton, Duc Nghia Pham, Stuart Bain, 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, AAAI Press/The MIT Press, 2004, pp. 191–196.
[53] Dave A.D. Tompkins, Holger H. Hoos, UBCSAT: an implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT, in: Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing, SAT, Vancouver, BC, Canada, 2004, pp. 37–46.
[54] Darrell Whitley, Adele E. Howe, Doug Hains, Greedy or not? Best improving versus first improving stochastic local search for MAXSAT, in: Proceedings of the 27th AAAI Conference on Artificial Intelligence, AAAI, Bellevue, Washington, USA, 2013, pp. 940–946.
[55] Shlomo Zilberstein, Using anytime algorithms in intelligent systems, AI Mag. 17 (3) (1996) 73–83.