Tailoring Local Search for Partial MaxSAT

Shaowei Cai[1,2*], **Chuan Luo**[3], **John Thornton**[4], **Kaile Su**[4]
[1]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2]Queensland Research Laboratory, NICTA, Brisbane, Australia
[3]Key Laboratory of High Confidence Software Technologies, Peking University, Beijing, China
[4]Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
shaoweicai.cs@gmail.com; chuanluosaber@gmail.com; {j.thornton,k.su}@griffith.edu.au

Abstract

Partial MaxSAT (PMS) is a generalization to SAT and MaxSAT. Many real world problems can be encoded into PMS in a more natural and compact way than SAT and MaxSAT. In this paper, we propose new ideas for local search for PMS, which mainly rely on the distinction between hard and soft clauses. We use these ideas to develop a local search PMS algorithm called *Dist*. Experimental results on PMS benchmarks from MaxSAT Evaluation 2013 show that *Dist* significantly outperforms state-of-the-art PMS algorithms, including both local search algorithms and complete ones, on random and crafted benchmarks. For the industrial benchmark, *Dist* dramatically outperforms previous local search algorithms and is comparable with complete algorithms.

Introduction

The maximum satisfiability problem (MaxSAT) is the optimization version of the satisfiability problem (SAT) which consists in finding an assignment that maximizes the number of satisfied clauses. A significant extension of MaxSAT is the Partial MaxSAT (PMS) problem, in which clauses are divided into hard and soft and the goal is to find an assignment that satisfies all hard clauses and the maximum number of soft clauses.

Combinatorial 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. Applications of PMS include network routing (Jiang, Kautz, and Selman 1995), scheduling problems (Thornton and Sattar 1998), timetabling problems (Cha et al. 1997), combinatorial auctions, and FPGA routing (Fu and Malik 2006), etc. Moreover, 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.

PMS (as with MaxSAT) is an NP-hard problem, and practical algorithms for this problem mainly fall into two types: complete algorithms and incomplete algorithms. Complete algorithms guarantee the optimality of the solutions they find, but may fail to find a good solution within reasonable time for large instances. Incomplete algorithms, mainly including local search algorithms, cannot guarantee the optimality of their solutions, but they can find optimal or near-optimal solutions within reasonable time, and thus are particularly useful in real world applications where the requirement is to find a good approximate solution quickly.

*Corresponding author

Related Works

There are numerous complete algorithms for solving PMS. A large family of complete PMS algorithms employ a branch and bound algorithm scheme (Li, Manyà, and Planes 2007; Lin, Su, and Li 2008; Heras, Larrosa, and Oliveras 2008; Li et al. 2009; Davies, Cho, and Bacchus 2010). Another large family of complete PMS algorithms are based on iteratively calling a SAT solver (Fu and Malik 2006; Berre and Parrain 2010; Koshimura et al. 2012; Ansótegui, Bonet, and Levy 2013; Martins, Manquinho, and Lynce 2013; Morgado et al. 2013). 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 Program (ILP) and solve the instance by a Mixed Integer Programming (MIP) solver (Ansótegui and Gabàs 2013).

Local search solvers 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. There have been numerous works on local search for MaxSAT, and a survey can be found in (Hoos and Stützle 2005). Also, local search techniques for SAT can be applied to MaxSAT. Most earlier successful local search algorithms for SAT have been extended for approximating MaxSAT in the UBCSAT system (Tompkins and Hoos 2004). Recently, a local search MaxSAT algorithm called *CCLS* won four categories in the incomplete track of MaxSAT Evaluation 2013, thanks to the configuration checking strategy (Cai, Su, and Sattar 2011), which has shown success in SAT solving (Cai and Su 2013). However, local search MaxSAT solvers do not show good performance on PMS instances, especially on non-random ones, when compared to complete solvers.

There are also efforts devoted to specialized local search

algorithms for PMS. Unlike the situation of MaxSAT, local search algorithms for PMS do not benefit that much from the techniques for SAT, mainly due to its special features. To develop effective local search PMS algorithms, 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 (Jiang, Kautz, and Selman 1995), which also tackles PMS as weighted MaxSAT (with the encoding mentioned above), 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 (Cha et al. 1997). 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) (Cha et al. 1997). This was further improved by Thornton et. al. by maintaining a dynamic weight differential between hard and soft clauses (Thornton and Sattar 1998; Thornton et al. 2002), resulting in the *TWO-LEVEL* algorithm. Experimental results in (Thornton and Sattar 1998) showed the superiority of dynamic weighting strategies over the fixed weighting strategy in (Cha et al. 1997).

Main Contributions and Structure of the Paper

In this work, we propose new ideas for local search for PMS. Based on these ideas, we develop a local search algorithm dubbed *Dist*, as it makes good use of *Dist*inctions between hard and soft clauses.

We propose a clause weighting scheme which only works on hard clauses. Moreover, unlike previous PMS local search algorithms, we separate hard score and soft score, and design a variable selection heuristic accordingly. The heuristic first prefers to satisfy more hard clauses, and then tries to satisfy more soft clauses without increasing number of falsified hard clauses. When neither case is possible, a variable from a falsified clause is selected, based on the best soft score criterion.

We compare *Dist* with state-of-the-art solvers including both local search solvers and complete ones on all PMS benchmarks from MaxSAT Evaluation 2013. Experimental results show that *Dist* outperforms all other solvers on random and crafted benchmarks. For the industrial benchmark, *Dist* dramatically outperforms previous local search algorithms and is comparable to complete algorithms. To the best of our knowledge, this is the first local search algorithm that is comparable to complete algorithms on industrial PMS instances.

The remainder of this paper is organized as follows: some preliminary concepts are given in the next section. Then we introduce the main new ideas in our algorithm and propose the *Dist* algorithm. After that, we present experimental results demonstrating the performance of *Dist* and study the effectiveness of the algorithmic components of *Dist*. Finally, we give some concluding remarks and future directions.

Preliminaries

Given a set of n Boolean $variables$ $\{x_1, x_2, ..., x_n\}$, a $literal$ is either a variable x_i or its negation $\neg x_i$, and a $clause$ is a disjunction of literals. A Conjunctive Normal Form (CNF) formula is a conjunction of clauses, and can be expressed as a set of clauses. 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). PMS is the SAT problem when there are no soft clauses, and is the MaxSAT problem when there are no hard clauses. In this sense, PMS is a generalization to both SAT and MaxSAT.

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 the minimum cost. The basic schema for local search algorithms for PMS (as with MaxSAT) is as follows. Starting with a randomly generated complete assignment, the algorithm chooses a variable and flips it in each subsequent step, trying to find a feasible assignment with a lower cost.

Main Ideas

In this section, we present new ideas in our local search algorithm for solving PMS.

Weighting Only for Hard Clauses

Our algorithm employs a clause weighting scheme that works only on hard clauses. This is essentially different from previous local search PMS algorithms which also utilize clause weighting schemes, as they increase weights of all falsified clauses, including both hard and soft ones.

Our weighting scheme works as follows. 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, hard clause weights are updated according to a weighting scheme similar to PAWS (Thornton et al. 2004): 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.

Some intuitive explanations behind the idea of weighting only for hard clauses are presented below.

Why 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

[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.

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 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.

Why not weighting for soft clauses: Some soft clauses might be usually falsified in local optima, due to a 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.

Separating Hard Score and Soft Score

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

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 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 and the unweighted version of soft score, as it employs a clause weighting scheme that works only for hard clauses.

To facilitate our discussions about *Dist* afterwards, we give some definitions here. In local search algorithms for SAT and MaxSAT, a variable is said to be decreasing 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 $A \times hscore(x) + sscore(x)$, 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 PMS algorithms is how to control the value of A to make the search more effective (Cha et al. 1997; Thornton and Sattar 1998; Thornton et al. 2002).

However, no matter which 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.

In contrast to previous local search algorithms, *Dist* utilizes hard score and soft score separately. 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.

Variable Selection Based on Hard and Soft Scores

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 to be flipped.

   However, there is still one question that needs to be answered: when there is more than one hard-decreasing variable, which one should be selected? Two natural answers are to pick the best one or to pick one randomly. It is difficult to decide which one is the best, unless there is some variable whose $hscore$ and $sscore$ are both the greatest, which rarely happens. On the other hand, randomized strategies are more robust and thus is adopted in *Dist* (with bias towards the one with the best $hscore$).

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 are present. Specifically, the one with the best positive $sscore$ is picked, which seems the 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. So, 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 (Jiang, Kautz, and Selman 1995)).

A question is then which variable should be chosen from the selected clause. Since before the stuck situation we mainly focus on satisfying hard clauses or at least protecting them, what we need here should be a disturbance of 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 helping to satisfy as many soft clauses as possible.

The *Dist* Algorithm

Based on the ideas in the preceding section, we develop an efficient local search algorithm for solving PMS, which is called *Dist* as it makes good use of the *Dist*inctions between hard and soft clauses. The *Dist* algorithm is outlined in Algorithm 1, as described below.

In the beginning, *Dist* generates an assignment α randomly, and the cost of the best feasible solution, denoted by $cost^*$, is initialized as $+\infty$. After the initialization, the loop (lines 3-15) is executed until a limited number of steps 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, *Dist* picks a random hard-decreasing variable (i.e., $hscore(x) > 0$) if any. If no hard-decreasing variable is present, then *Dist* picks a best variable (w.r.t. $sscore$) from the set S of variables with 0-$hscore$ and positive $sscore$. If there do not exist hard-decreasing variables and the S set is empty, which means the algorithm gets stuck, then *Dist* updates hard clause weights according to the weighting 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 any, and from falsified soft clauses otherwise. The variable with the greatest $sscore$ from the chosen falsified clause is selected. Note that in *Dist*, all ties are broken randomly. In practice, in order to make the algorithm more robust, we also employ a pure random walk step with a small probability in each random step, as suggested in (Hoos 1999).

Finally, when the loop terminates on reaching the step or 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".

Also note that *Dist* is a specialized algorithm for solving partial MaxSAT, and when solving MaxSAT instances containing only soft clauses, it can be seen as an extension of GSAT (Selman, Levesque, and Mitchell 1992).

Experimental Evaluation

We empirically evaluate *Dist* on all Partial MaxSAT benchmarks in MaxSAT Evaluation 2013, including three categories namely random, crafted and industrial. We compare *Dist* with state-of-the-art local search algorithms and the best complete algorithms for these benchmarks.

Algorithm 1: *Dist*

Input: Partial MaxSAT instance F, $maxSteps$
1 $\alpha :=$ randomly generated truth assignment;
2 $cost^* := +\infty$;
3 **for** $step := 1$ **to** $maxSteps$ **do**
4 **if** $\nexists$ *falsified hard clauses* & $cost(\alpha) < cost^*$ **then**
5 $\alpha^* := \alpha$; $cost^* := cost(\alpha)$;
6 **if** $H := \{x | hscore(x) > 0\} \neq \emptyset$ **then**
7 $v :=$ a variable randomly picked from H;
8 **else if** S:=$\{x | hscore(x)=0$ & $sscore(x)>0\} \neq \emptyset$ **then**
9 $v :=$ a variable in S with the greatest $sscore$;
10 **else**
11 update weights of hard clauses;
12 **if** $\exists$ *falsified hard clauses* **then** $c :=$ a random falsified hard clause;
13 **else** $c :=$ a random falsified soft clause;
14 $v :=$ a variable in c with the greatest $sscore$;
15 $\alpha := \alpha$ with v flipped;
16 **if** $cost^* \leq$ *#(soft clauses)* **then return** $(cost^*, \alpha^*)$;
17 **else return** "no feasible assignment found";

Experimental Setup

Dist is implemented in C++ and complied by g++ with the '-O2' option, and will be distributed online. The sp parameter for the weighting scheme is set to 0.001[2].

We compare *Dist* with two local search solvers, *TWO-LEVEL* and *CCLS*. While *TWO-LEVEL* (Thornton et al. 2002) is a state-of-the-art PMS local search solver, *CCLS* is a local search solver for weighted MaxSAT that won random and crafted weighted MaxSAT categories in the incomplete track of MaxSAT Evaluation 2013. *CCLS* employs a diversification strategy called configuration checking that has proved successful in SAT solving (Cai and Su 2012; Luo et al. 2013; Cai and Su 2013).

We also compare *Dist* with four complete solvers, including *WMaxSatz09* (Li et al. 2009), *ILP-2013* (Ansótegui and Gabàs 2013), *QMaxSAT2-mt* (Koshimura et al. 2012) and *optimax-it*. The first three are the best algorithms in the random, crafted and industrial Partial MaxSAT category of MaxSAT Evaluation 2013 respectively[3]. The solver *optimax-it* was slightly modified from the complete solver optimax and won all Partial MaxSAT categories and the industrial MaxSAT category in the incomplete track. Note that we do not compare *Dist* with the portfolio solver *ISAC+-pms* (Ansotegui, Malitsky, and Sellmann 2014) despite the fact it performs better than the above complete algorithms. The reason is that *ISAC+-pms* is a collection of algorithms, and it calls different algorithms depending on the input instance.

Interestingly, these complete solvers belong to different types of methods. *WMaxSatz09* is a branch-and-bound algorithm; *ILP-2013* converts MaxSAT into ILP and uses the MIP solver IBM-CPLEX to solve the ILP instances; *QMaxSAT2-mt* and *optimax-it* are SAT-based solvers.

[2] except that 0.0001 for industrial instances with more than 2500 variables

[3] http://maxsat.ia.udl.cat:81/13/results/index.html

Instance set	#	*Dist*	*CCLS*	*WMaxSatz09*	*TWO-LEVEL*	*ILP-2013*	*QMaxSAT2-mt*	*optimax-it*
min2sat/v160c800l2	30	**0.13(30)**	0.38(30)	153.08(27)	691.89(2)	225.65(26)	0(0)	0(0)
min2sat/v260c1040l2	30	**0.67(30)**	1.44(30)	133.96(18)	375.10(4)	171.53(21)	0(0)	0(0)
min3sat/c70v350l3	30	**0.02(30)**	0.12(30)	134.75(30)	15.94(30)	525.46(13)	350.68(2)	0(0)
min3sat/c80v400l3	30	**0.02(30)**	0.09(30)	332.23(26)	128.73(28)	0(0)	0(0)	0(0)
pmax2sat/hi	29	**0.01(29)**	5.67(29)	7.75(29)	0.07(28)	308.34(2)	0(0)	0(0)
pmax2sat/me	30	**0.01(30)**	0.48(29)	3.31(30)	0.04(30)	173.08(12)	0(0)	0(0)
pmax3sat/hi	30	**<0.01(30)**	0.01(30)	28.42(30)	0.01(30)	615.04(7)	214.04(1)	0(0)
Total solved	209	**209**	208	190	152	81	3	0
Mean ratio		**100%**	99.52%	90.95%	72.84%	38.60%	1.43%	0%

Table 1: Experimental results on random PMS benchmark from MaxSAT Evaluation 2013. We remove one instance from the "pmax2sat/hi" family as it has been proved that the hard clauses compose an unsatisfiable formula and thus no feasible assignment exists for the instance.

Instance set	#	*Dist*	*ILP-2013*	*WMaxSatz09*	*QMaxSAT2-mt*	*TWO-LEVEL*	*CCLS*	*optmax-it*
frb	25	**141.16(23)**	214.56(11)	111.85(5)	15.81(22)	79.57(5)	34.42(17)	106.1(3)
job-shop	3	0(0)	0(0)	0(0)	**20.55(3)**	0(0)	0(0)	43.77(3)
maxclicque/random	96	**0.03(96)**	31.12(96)	0.69(96)	20.32(80)	0.09(92)	0.24(96)	67.23(77)
maxclicque/structured	62	**30.95(59)**	125.48(38)	41.78(39)	26.87(28)	7.15(39)	25.56(43)	33.78(22)
maxone/3sat	80	**0.18(80)**	4.93(80)	0.22(80)	67.31(71)	61.43(70)	1.84(75)	34.77(64)
maxone/structured	60	46.55(56)	67.22(58)	43.22(59)	**8.61(60)**	12.32(36)	124.57(1)	1.59(52)
min-enc/kbtree	42	**9.36(42)**	125.71(42)	373.20(20)	71.25(6)	43.53(33)	24.94(39)	60.47(3)
pseudo/miplib	4	**0.02(4)**	4.93(4)	0.61(4)	0.05(4)	150.39(3)	0.03(4)	0.47(4)
scheduling	5	211.23(1)	0(0)	0(0)	**560.67(4)**	0(0)	0(0)	0(0)
Toal solved	377	**361**	329	303	279	278	275	228
Mean ratio		**77.83%**	66.88%	58.76%	77.72%	53.33%	58.40%	55.72%

Table 2: Experimental results on crafted Partial MaxSAT benchmark from MaxSAT Evaluation 2013.

All experiments are carried out on a workstation under Ubuntu Linux operation system, using an Intel(R) Core(TM) i7-2640M 2.8 GHz CPU and 8 GB RAM. The time limit is set to be 1000 CPU seconds for each run.

Evaluation Methodology

We follow the evaluation methodology adopted in the incomplete track of MaxSAT evaluations: Each solver is executed once on each instance, where the solver prints successively the best solution it finds so far. For each solver on each instance family, we report within parenthesis the number of instances where the solver finds the best solution, and the mean time of doing so over such "winning" instances (not including proving time for complete solvers). Solvers are ordered from left to right according to the total number of "winning" instances. The number of instances of each family is specified in the column "#". Since the number of instances varies among families, we also present for each solver the mean ratio of "winning" instances. The rules at 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.

Experimental Results

In this subsection, we present the comparative experimental results of *Dist* and its competitors on each benchmark.

Results on Random PMS Benchmark:

Table 1 shows the comparative results on the random PMS benchmark. As is clear from Table 1, *Dist* dominates on all instance families. The only other solver that might be comparable to *Dist* on random instances is *CCLS*. So we take a further look at the comparison between *Dist* and *CCLS*. We observe that *Dist* is more robust and more efficient than *CCLS*. *Dist* finds the best solution for all instances very quickly, while *CCLS* fails to find the best solution for one partial Max-2-SAT instance. Also, the averaged run time over all successful instances (to find the best solution) of *Dist* is 10 times less than that of *CCLS*.

Results on Crafted PMS Benchmark:

The experimental results on the crafted PMS benchmark are presented in Table 2. Among 377 instances, *Dist* finds the best solution for 361 of them, more than any other solver. In detail, *Dist* gives the best performance for 6 out of 9 instance families, while the other 3 families are dominated by the complete solver *QMaxSAT2-mt*. We would like to note that *Dist* is the only local search solver that performs better than complete solvers on crafted instances.

Results on Industrial PMS Benchmark:

Table 3 summarizes the experimental results on the industrial PMS benchmark. When comparing the local search solvers, *Dist* shows substantial superiority over the other local search solvers. *Dist* finds the best solution for 337 instances, while this figure is only 80 and 70 for *CCLS* and *TWO-LEVEL* respectively.

For this industrial benchmark, the two SAT-based complete solvers *QMaxSAT2-mt* and *optmax-it* are the best two solvers for these industrial instances. Nevertheless, *Dist* shows better performance than other types of complete solvers, including the ILP-translation approach *ILP-2013* and the branch-and-bound algorithm *WMaxSatz09*. We also observe that, among the four instance families

Instance set	#	*QMaxSAT2-mt*	*optmax-it*	*Dist*	*ILP-2013*	*WMaxSatz09*	*CCLS*	*TWO-LEVEL*
aes	7	0.72(1)	100.94(2)	**5.01(6)**	18.22(2)	0(0)	1.41(4)	0.01(1)
bcp-fir	50	**26.36(49)**	13.6(48)	92.64(37)	9.66(50)	213.63(28)	91.49(25)	0(0)
bcp-hipp-yRa1/simp	17	**6.54(17)**	2.67(16)	14.49(17)	1.27(5)	1.49(6)	1.65(7)	0(0)
bcp-hipp-yRa1/su	38	**28.83(38)**	8.68(29)	104.33(32)	0(0)	0(0)	0(0)	0(0)
bcp-msp	50	41.11(35)	17.35(14)	**56.87(41)**	142.02(25)	95.27(23)	0(0)	11.88(20)
bcp-mtg	40	**0.11(40)**	0.14(40)	181.03(36)	88.29(24)	108.31(14)	0(0)	0(0)
bcp-syn	50	50.98(23)	20.46(25)	65.22(28)	9.73(48)	48.62(22)	0.44(20)	71.17(19)
trial/circuit-trace-compaction	4	**5.62(4)**	49.9(4)	0(0)	318.81(1)	57.94(1)	0(0)	0(0)
close_solutions	50	**85.41(50)**	17.55(37)	53.11(38)	123.84(30)	0(0)	47.41(11)	27.66(14)
des	50	**217.02(50)**	37.91(20)	779.53(12)	247.39(16)	0(0)	0(0)	930.18(1)
haplotype-assembly	6	**49.29(6)**	0.42(4)	0(0)	229.55(3)	0(0)	0(0)	0(0)
packup-pms	40	**4.17(40)**	1.85(40)	142.72(12)	0.66(40)	0(0)	0(0)	0(0)
pbo-mqc/nencdr	50	**22.17(50)**	23.91(42)	0(0)	266.59(1)	288.47(23)	0(0)	0(0)
pbo-mqc/nlogencdr	50	**11.32(50)**	19.98(48)	379.14(14)	238.11(2)	188.24(38)	0(0)	0(0)
pbo-routing	15	1.76(15)	**0.23(15)**	15.98(15)	21.03(15)	2.01(5)	0(0)	13.94(14)
protein_ins	12	10.35(12)	170.68(10)	**1.99(12)**	0.15(1)	0.07(2)	4.09(12)	0(0)
tpr/Multiple_path	48	**51.68(48)**	167.13(37)	59.42(12)	61.13(14)	374.45(5)	0(0)	0(0)
tpr/One_path	50	**12.89(50)**	43.79(46)	9.18(25)	23.32(50)	3.15(25)	0(0)	0(0)
Total	627	**582**	478	337	327	192	80	70
Mean ratio		**90.46%**	77.01%	55.83%	48.58%	26.32%	17.35%	12.31%

Table 3: Experimental results on industrial Partial MaxSAT benchmark from MaxSAT Evaluation 2013.

which are not dominated by *QMaxSAT2-mt*, *Dist* gives the best performance for 3 of them. In this sense, *Dist* could be complementary to SAT-based solvers. It would be interesting to underpin this assumption by using a spearman correlation test or the marginal contribution to the oracle performance (Xu et al. 2012). We leave this for future work.

Discussion

In this section, we study the effectiveness of the weighting scheme as well as two strategies in *Dist*.

We compare *Dist* with its three alternatives, which are modified from *Dist* as follows.

- *Dist*_alt1: update all clause weights (replace line 11 in Algorithm 1);
- *Dist*_alt2: when there are hard-decreasing variables, pick the one with the best $hscore$, breaking ties by preferring the one with the greatest $sscore$. (replace line 7 in Algorithm 1);
- *Dist*_alt3: pick a random variable from the selected falsified clause (replace line 14 in Algorithm 1).

Benchmark	#	*Dist*	*Dist*_alt1	*Dist*_alt2	*Dist*_alt3
Random	209	**209**	59	89	**209**
Crafted	377	**361**	317	329	294
Industrial	627	**337**	242	246	202

Table 4: Experimental results comparing *Dist* with its alternatives. For each benchmark, each cell reports for each solver the number of instances where it finds the best solution.

The experimental results (Table 4) demonstrate the superiority of *Dist* over its alternatives, which indicates the effectiveness of the algorithmic components. We also note that these degenerated alternatives of *Dist* still exhibit significantly better performance than previous local search solvers on crafted and industrial benchmarks. This indicates the algorithmic framework of *Dist*, which is based on separated hard score and soft score, is more promising than previous local search approaches on structured PMS instances.

Conclusions and Future Work

In this work, we proposed 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, and a variable selection heuristic which utilizes hard score and soft score separately. 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, and is comparable and complementary on industrial instances.

It is natural to extend our method to weighted partial MaxSAT. Also, the strong experimental results suggest that local search based on hard and soft score is a promising direction for solving PMS and deserves further research. We consider to improve *Dist* by designing more effective heuristics. For example, when there are hard-decreasing variables, we can utilize more sophisticated variable selection strategies, such as considering the dominating variable set of hard-decreasing variables.

Acknowledgement

This work is supported by 973 Program 2010CB328103, ARC Grant FT0991785, and NSFC 6137007.

References

Ansótegui, C., and Gabàs, J. 2013. Solving (weighted) partial MaxSAT with ILP. In *Proc. of CPAIOR-13*, 403–409.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2013. SAT-based MaxSAT algorithms. *Artif. Intell.* 196:77–105.

Ansotegui, C.; Malitsky, Y.; and Sellmann, M. 2014. Maxsat portfolio. In *Proc. of AAAI-2014*, to appear.

Berre, D. L., and Parrain, A. 2010. The sat4j library, release 2.2. *JSAT* 7(2-3):59–64.

Cai, S., and Su, K. 2012. Configuration checking with aspiration in local search for SAT. In *Proc. of AAAI-12*, 334–340.

Cai, S., and Su, K. 2013. Local search for Boolean Satisfiability with configuration checking and subscore. *Artif. Intell.* 204:75–98.

Cai, S.; Su, K.; and Sattar, A. 2011. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artif. Intell.* 175(9-10):1672–1696.

Cha, B.; Iwama, K.; Kambayashi, Y.; and Miyazaki, S. 1997. Local search algorithms for partial MAXSAT. In *Proc. of AAAI/IAAI-97*, 263–268.

Davies, J.; Cho, J.; and Bacchus, F. 2010. Using learnt clauses in MaxSAT. In *Proc. of CP-10*, 176–190.

Fu, Z., and Malik, S. 2006. On solving the partial MAX-SAT problem. In *Proc. of SAT-06*, 252–265.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. Minimaxsat: An efficient weighted Max-SAT solver. *J. Artif. Intell. Res.* 31:1–32.

Hoos, H. H., and Stützle, T. 2005. *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann.

Hoos, H. H. 1999. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proc. of AAAI/IAAI-99*, 661–666.

Jiang, Y.; Kautz, H.; and Selman, B. 1995. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In *First Interational Joint Workshop on Artificial Intelligence and Operations Research*.

Koshimura, M.; Zhang, T.; Fujita, H.; and Hasegawa, R. 2012. Qmaxsat: A partial Max-SAT solver. *JSAT* 8(1/2):95–100.

Li, C. M.; Manyà, F.; Mohamedou, N. O.; and Planes, J. 2009. Exploiting cycle structures in Max-SAT. In *Proc. of SAT-09*, 467–480.

Li, C. M.; Manyà, F.; and Planes, J. 2007. New inference rules for Max-SAT. *J. Artif. Intell. Res.* 30:321–359.

Lin, H.; Su, K.; and Li, C. M. 2008. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *Proc. of AAAI-08*, 351–356.

Luo, C.; Cai, S.; Wu, W.; and Su, K. 2013. Focused random walk with configuration checking and break minimum for satisfiability. In *Proc. of CP-13*, 481–496.

Martins, R.; Manquinho, V. M.; and Lynce, I. 2013. Community-based partitioning for MaxSAT solving. In *Proc. of SAT-13*, 182–191.

Morgado, A.; Heras, F.; Liffiton, M. H.; Planes, J.; and Marques-Silva, J. 2013. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints* 18(4):478–534.

Selman, B.; Levesque, H. J.; and Mitchell, D. G. 1992. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92*, 440–446.

Thornton, J., and Sattar, A. 1998. Dynamic constraint weighting for over-constrained problems. In *Proc. of PRICAI-98*, 377–388.

Thornton, J.; Bain, S.; Sattar, A.; and Pham, D. N. 2002. A two level local search for MAX-SAT problems with hard and soft constraints. In *Proc. of AUS-AI-02*, 603–614.

Thornton, J.; Pham, D. N.; Bain, S.; and Ferreira Jr., V. 2004. Additive versus multiplicative clause weighting for SAT. In *Proc. of AAAI-04*, 191–196.

Tompkins, D. A. D., and Hoos, H. H. 2004. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In *Proc. of SAT-04*, 37–46.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2012. Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proc. of SAT-12*, 228–241.