From Decimation to Local Search and Back: A New Approach to MaxSAT

Shaowei Cai[1,2*], Chuan Luo[3], Haochen Zhang[1]
[1]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[2]School of Computer and Control Engineering, University of Chinese Academy of Sciences, China
[3]Institute of Computing Technology, Chinese Academy of Sciences, China
caisw@ios.ac.cn; chuanluosaber@gmail.com; femrat@gmail.com

Abstract

Maximum Satisfiability (MaxSAT) is an important NP-hard combinatorial optimization problem with many applications and MaxSAT solving has attracted much interest. This work proposes a new incomplete approach to MaxSAT. We propose a novel decimation algorithm for MaxSAT, and then combine it with a local search algorithm. Our approach works by interleaving between the decimation algorithm and the local search algorithm, with useful information passed between them. Experiments show that our solver DeciLS achieves state of the art performance on all unweighted benchmarks from the MaxSAT Evaluation 2016. Moreover, compared to SAT-based MaxSAT solvers which dominate industrial benchmarks for years, it performs better on industrial benchmarks and significantly better on application formulas from SAT Competition. We also extend this approach to (Weighted) Partial MaxSAT, and the resulting solvers significantly improve local search solvers on crafted and industrial benchmarks, and are complementary (better on WPMS crafted benchmarks) to SAT-based solvers.

1 Introduction

Given a propositional formula expressed in the Conjunctive Normal Form (CNF), the maximum satisfiability problem (MaxSAT) is to find an assignment of Boolean values to the variables of the formula that maximizes the number of satisfied clauses (equally minimizes the number of unsatisfied clauses). MaxSAT is the optimization version of the famous NP-complete problem namely the propositional satisfiability problem (SAT), and has applications in various domains [Xu *et al.*, 2003; Safarpour *et al.*, 2007; Chen *et al.*, 2010; Janota *et al.*, 2012]. An important variant 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.

There has been much interest in developing efficient MaxSAT solvers, including both complete solvers and incomplete solvers. A MaxSAT solver is complete if it proves the optimality of the solution it finds when it terminates. Otherwise, it is an incomplete solver. Over the past decade, the most successful approach to solving industrial instances of MaxSAT is the SAT-based approach, which relies on iteratively calling a SAT solver.

For many large industrial instances, because of the NP-hardness of the problem and the time constraint, we do not expect to solve them exactly. Instead, the goal is often to find a good-quality solution in a reasonable amount of time. To this end, many incomplete algorithms have been developed.

A main incomplete approach to MaxSAT is local search. Typically, a local search algorithm for MaxSAT starts with a complete assignment, and chooses a variable and flips its value in each subsequent step. The best found assignment is returned when the algorithm reaches its termination condition (e.g., a time limit). Modern local search solvers for MaxSAT [Luo *et al.*, 2015; Cai *et al.*, 2016; Luo *et al.*, 2017] prove efficient in solving random and crafted instances, but their performance on industrial instances is poor.

Another incomplete approach of interest is the decimation-based approach, which proceeds by assigning sequentially the value of some (typically one) of the variables and simplifies the formula accordingly. A main kind of decimation algorithms for satisfiable SAT problems are the message passing algorithms, such as the Survey Propagation (SP) algorithm [Braunstein *et al.*, 2005]. SP has been extended to solve MaxSAT in [Chieu and Lee, 2009]. These algorithms have close connections to statistical physics, and they can be hardly applied to structured instances, mainly because they rely heavily on the statistical properties of solutions [Altarelli *et al.*, 2009]. There are also decimation algorithms based on unit propagation (UP) [Chao and Franco, 1986; 1990; Paturi *et al.*, 2005], which have been studied in the context of SAT and mainly for theoretical interest.

Although various incomplete approaches have been developed for solving MaxSAT, they usually have poor performance on industrial instances. On the other hand, SAT-based complete MaxSAT solvers have achieved very good performance on a wide range of industrial instances. This motivates researchers to modify the SAT-based solvers to make them incomplete (i.e., by returning better assignments when found) [Ansótegui *et al.*, 2016]. Such incomplete solvers dominate the industrial categories of the incomplete

*Corresponding author

track in recent MaxSAT Evaluations (MSEs).

This work proposes a new incomplete approach to efficiently solving MaxSAT. Firstly, we propose a UP-based decimation algorithm for MaxSAT, which, to the the best of our knowledge, is the first UP-based decimation algorithm for MaxSAT. UP-based decimation algorithms have been studied for SAT, and they fail when a conflict is detected. However, when solving MaxSAT, it is unreasonable to stop the decimation procedure upon detecting a conflict, since there are usually unsatisfied clauses under optimal assignments. Instead, in this case, we introduce a heuristic to assign a value to the variable to which the two contradictory unit clause corresponds. When there is no unit clause, we also use the heuristic to assign variables until UP can be applied again.

By combining this decimation algorithm with an existing local search algorithm, we design a solver for MaxSAT named DeciLS.The solver works by interleaving between a decimation procedure and a local search procedure. The assignment produced by the decimation procedure is fed as the initial assignment to local search. More importantly, a feedback mechanism is proposed to provide the best assignment found by the local search procedure to guide the next decimation procedure on assigning variables when UP cannot be applied. We carry out experiments to compare DeciLS against state of the art MaxSAT solvers on all unweighted benchmarks from the MSE 2016 and application formulas from the SAT Competition 2016. The results show that DeciLS has the best performance on all benchmarks except the crafted one. In particular, it outperforms a state of the art SAT-based solver WPM3-2015-in and a recent portfolio solver on all benchmarks, and performs better on industrial benchmark from MSE 2016 and significantly better on the application formulas from the SAT Competition 2016.

Furthermore, we extend our approach to PMS and WPMS, by exploiting the distinction of hard and soft clauses and addressing the issue of no feasible solution found by local search. We implement it with state of the art local search solvers, leading to two hybrid incomplete solvers. Our experiments show that their performance on (W)PMS crafted and industrial benchmarks is significantly better than local search solvers and is complementary and sometime competitive to SAT-based solvers.

The reminder of this paper is organized as follows. The next section introduces the notation and definitions used in the paper. In Section 3, we discuss the related works. We propose our approach in Section 4, and present experimental results of our solver DeciLS in Section 5. Then in Section 6, we extend the approach to (W)PMS and develop two incomplete solvers named DeciDist and DeciCCEHC, and present related experiment results. Finally, we conclude the paper.

2 Preliminaries

Given a set of Boolean $variables$ $\{x_1, x_2, ..., x_n\}$, a $literal$ is either a variable x (which is called a positive literal) or its negation $\neg x$ (which is called a negative literal). A CNF formula F is a conjunction of clauses (i.e., $F = C_1 \wedge C_2 \wedge ... \wedge C_m$), where a $clause$ is a disjunction of literals (i.e., $C_i = \ell_{i1} \vee \ell_{i2} \vee ... \vee \ell_{ij}$). A clause that contains only one single literal is called a *unit clause*. For a formula F, we denote the set of variables in F by $Var(F)$.

For a formula F, an *assignment* α is a mapping $Var(F) \rightarrow \{0, 1\}$. A complete assignment is a mapping that assigns to each variable either 0 or 1. A clause is *satisfied* if it has at least one true literal, and *unsatisfied* if all the literals in the clause are false literals. By convention the empty clause is unsatisfied. The *cost* of an assignment α (may be partial), denoted by $cost(\alpha)$, is defined to be the number of unsatisfied clauses under α. Given a CNF formula, the MaxSAT problem can be regarded as finding a complete assignment with the minimum cost.

In the Partial MaxSAT (PMS) problem, some clauses are declared to be hard and the rest are declared to be soft, and the task is to find an assignment such that all hard clauses are satisfied and the number of unsatisfied soft clauses is minimized. 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 is defined to be the number of unsatisfied soft clauses under α. In Weighted PMS (WPMS), each soft clause is associated with an integer as its weight, and the cost of a feasible assignment is defined to be the total weight of unsatisfied soft clauses.

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 $\neg\ell$ by the constant false, and simplifying accordingly. The result of conditioning F on ℓ is denoted by $F|_\ell$ and can be described succinctly as follows: $F|_\ell = \{c/\{\neg\ell\} | c \in F, \ell \notin c\}$ [Darwiche and Pipatsrisawat, 2009]. Note that $F|_\ell$ does not contain any literal ℓ or $\neg\ell$. When we assign a variable x with a value v, we can simplify the formula accordingly.

Unit propagation on a CNF formula F works as follows: For a unit clauses in F, the variable is assigned to satisfy this unit clause. That is, if the unit clause $\{x_i\}$ appears in the formula, x_i is assigned to true; and if the unit clause $\{\neg x_i\}$ appears in the formula, x_i is assigned to false. Then the formula is conditioned on this setting. This is a *unit propagation step*. The iterative execution of such steps until no more unit clause remains is called *unit propagation* (UP).

3 Related Works

We are not aware of any UP-based decimation algorithm for MaxSAT. Nevertheless, there are a few decimation algorithms for SAT based on unit propagation, which are mainly studied for theoretical interest. An early example is an algorithm called Unit Clause (UC) [Chao and Franco, 1986], which works iteratively until all clauses are satisfied or a conflict is generated. At each step of UC, if there exist unit clauses, a unit clause is picked to perform unit propagation; otherwise, an unassigned variable is selected uniformly at random and is assigned to 1 or 0 uniformly at random. The UC algorithm was extended to the Generalized Unit Clause (GUC) algorithm, where the unassigned variable is always chosen among those appearing in the shortest clauses [Chao and Franco, 1990]. The core of an advanced theoretical SAT algorithm dubbed PPSZ [Paturi *et al.*, 2005] is a UP-based decimation

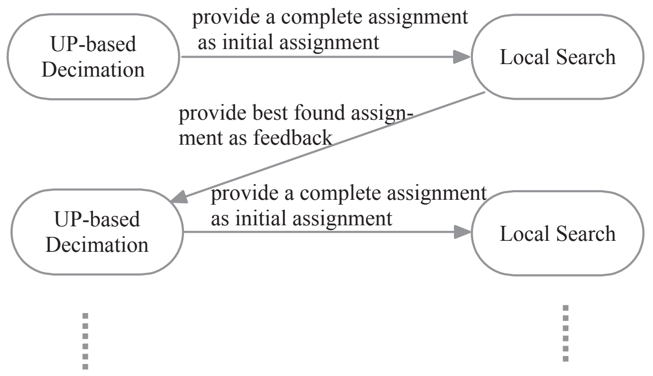
UP-based Decimation
provide a complete assignment as initial assignment
Local Search
provide best found assignment as feedback
UP-based Decimation
provide a complete assignment as initial assignment
Local Search

Figure 1: The process of our approach in a high level.

procedure as follows. Take a random order of variables and gradually assign values to them in this order. If a variable currently considered occurs in a unit clause, it is assigned to the value so that the clause is satisfied; otherwise, it is assigned randomly. At each step, if the formula is satisfied, the problem is solved. If two contradictory unit clauses are detected, the decimation procedure stops.

UP has been used in local search for SAT, and in these algorithms (e.g. UnitWalk [Hirsch and Kojevnikov, 2005]), UP is used during local search but not in the generation of initial assignment. Recently, UP was used in generating the initial assignment in local search solvers for PMS [Cai *et al.*, 2016]. The method simply uses UP to produce a partial assignment, and UP stops when no unit clause exists or a conflict is generated. After that, the resulting partial assignment is extended to a complete assignment by assigning all unassigned variables randomly. This initialization process was applied to a local search solver called Dist, leading to an improved solver called DistUP [Cai *et al.*, 2016].

4 A New Incomplete Approach to MaxSAT

In this work, we propose a new incomplete approach to MaxSAT, with the focus on solving industrial formulas. The proposed approach interleaves between a decimation algorithm and a local search algorithm, with message passing between each other. The process of our approach in a high level is illustrated in Figure 1, and is described below.

The approach works round by round. In each round, an execution of the decimation algorithm produces a complete assignment, which is then handed to the local search algorithm as the initial assignment for possible further improvement. Additionally, a feedback mechanism is employed to provide useful information from local search to the decimation algorithm of the next round. In our approach, the feedback information from local search to decimation refers to the best found assignment during local search of the previous round. However, it could be any information of interest, and the emphasis is on the feedback mechanism rather than the specific feedback information.

The feedback mechanism in our approach is described as follows. The best found assignment in the latest finished local search procedure, denoted by $prev\_ls\_best$, is recorded and maintained, and is used in the decimation procedure. In detail, in the UP-based decimation procedure, if UP cannot be applied, then the variable x under consideration (either it relates to a conflict or it does not appear in a unit clause) is assigned to $prev\_ls\_best[x]$.

Algorithm 1: A UP-based Decimation Algorithm for MaxSAT

```
Input: A CNF formula $F$
Output: An assignment of variables in $F$
1  while $\exists$ unassigned variables do
2      if $\exists$ $unit\ clauses$ then
3          if $\nexists$ $contradictory\ unit\ clauses$ then
4              $c \leftarrow$ pick a unit clause by pick_uc function;
5              perform a unit propagation step using $c$;
6          else
7              $x \leftarrow$ the variable related to the two
               contradictory unit clauses;
8              assign $x$ to $prev\_ls\_best[x]$, simplify $F$
               accordingly;
9      else
10         $x \leftarrow$ pick an unassigned variable by pick_var
           function;
11         assign $x$ to $prev\_ls\_best[x]$, simplify $F$
           accordingly;
12 return the resulting assignment to $Var(F)$;
```

4.1 The Decimation Algorithm for MaxSAT

In this subsection, we propose a UP-based decimation algorithm for MaxSAT, which is outlined in Algorithm 1. The algorithm works iteratively by assigning variables one by one.

In each step, if there exist unit clauses, the algorithm distinguishes two different cases. If there are not contradictory unit clauses, then it picks a unit clause to perform a unit propagation step. Otherwise, the variable x that relates to two contradictory unit clauses is assigned to the value $prev\_ls\_best[x]$, and the formula is simplified accordingly.

If there is no unit clause, then the algorithm picks an unassigned variable x and assigns it to the value $prev\_ls\_best[x]$. The formula is simplified accordingly.

As can be seen from Algorithm 1, we use the *pick_uc* function to pick a unit clause to perform UP, and the *pick_var* function to pick an unassigned variable to assign. Our idea is to employ assigning orders as distant as possible in different rounds of decimation. Firstly, this exploits different reason chains as many as possible, among which the correct (or nearly correct) one may be touched. We believe a key to efficiently solve industrial instances is to find the right reason chain. Secondly, different assigning orders help to generate assignments with large distances in the search space. These solutions are then improved by local search and thus can touch local optima in different areas. Additionally, recalling that our decimation algorithm consults to the best found solution in the last local search procedure, the introduction of this diversification is advisable.

With the above considerations, we utilize a heuristic for both the two functions, which is based on a diversification property. For a variable, we use $prev\_deci\_step[x]$ to denote the step number in which it was assigned in the previous round of decimation procedure. For two variables x and y, if $prev\_deci\_step[x] > prev\_deci\_step[y]$, it means that in the last round of decimation, x is assigned after y.

Our heuristic prefers to pick a variable with a large $prev\_deci\_step$ value. We use the Best from Multiple Selection (BMS) strategy [Cai, 2015] for this purpose. In detail, it chooses k variables (k is an integer parameter) randomly with replacement from the set of candidate variables, and returns the one with the greatest $prev\_deci\_step$ value. Note that to pick a unit clause among a set of unit clauses, we use the BMS strategy to select the variable from all variables corresponding to unit clauses, and the corresponding unit clause is returned.

4.2 The Local Search Solver

Our approach needs to call a local search solver. Instead of developing a solver from scratch, we modify a local search solver for SAT named CCAnr[1] to make it fit into our approach. As shown in [Cai *et al.*, 2015], CCAnr is amongst the best-performing local search solvers for structured SAT instances. The source code of CCAnr is available online.

In order to integrate CCAnr into our approach, we make a few modifications. Firstly, we add codes to keep track of the best found assignment in each round of local search procedure. Also note that, this data structure is accessible to the decimation algorithm. Secondly, we modify the initialization to make it receive the assignment from the decimation algorithm. Thirdly, each round of local search is stopped if the solution has not been improved in a fixed number of steps (this parameter is denoted as $MaxNonImprove$). Finally, we remove the ”Aspiration” mechanism to make it simpler, as removing it does not have obvious effect.

5 Experiment Evaluation of DeciLS

We conduct experiments to evaluate DeciLS on a broad range of unweighted MaxSAT benchmarks. Experiments show that DeciLS obtains state of the art performance on all benchmarks and performs better than state of the art for the industrial/application instances.

5.1 Experiment Preliminaries

Implementation: DeciLS is implemented in C++ and compiled by g++ with '-O3' option. There are two parameters in DeciLS: parameter k for the BMS heuristic used in the decimation algorithm and $MaxNonImprove$ for local search. We tune these parameters with the automatic configuration tool *SMAC* [Hutter *et al.*, 2011], and set $k = 20$ and $MaxNonImprove = 50000$ for all benchmarks.

Competitors: We compare our solver DeciLS with four state of the art MaxSAT solvers, including WPM3-2015-in [Ansótegui *et al.*, 2015], CCLS [Luo *et al.*, 2015], MiniWalk [Kroc *et al.*, 2009] and a recent portfolio solver dsat-wpm3-in-ms [Ansótegui *et al.*, 2017]. All these solvers except MiniWalk are the versions submitted to the MSE 2016. MiniWalk is provided by its authors.

[1]http://lcs.ios.ac.cn/ caisw/SAT.html

WPM3-2015-in is an incomplete version of the WPM3 solver [Ansótegui *et al.*, 2015]; it is the best SAT-based incomplete solver for the unweighted MaxSAT industrial category of the MSE2016, and placed 3rd in the category. The portfolio solver dsat-wpm3-in-ms placed 2nd in the category. Note that the winner of this industrial category is an implementation of DeciLS by us. CCLS represents state of the art in local search for unweighted MaxSAT, and has won several categories in recent MSEs, including the incomplete crafted category in MSE 2016. MiniWalk is a hybrid solver that uses a complete SAT solver MiniSAT [Eén and Biere, 2005] to guild a local search solver WalkSAT [Selman *et al.*, 1994] in parallel, and is effective for solving structured MaxSAT instances for which the optimal solutions have only a few unsatisfied clauses [Kroc *et al.*, 2009].

Additionally, to show the complimentary of the solvers, we also report the results of the Virtual Best Solver (*VBS*), i.e., the perfect selector (not an actual solver) – on each instance, the solution of *VBS* is the best one of the solutions reported by both solvers; if more than one solver report the best-quality solution, the computing time of *VBS* is the shortest one.

Benchmarks: We evaluate DeciLS on all random, crafted and industrial benchmarks of unweighted MaxSAT from the MSE2016. Also, we conduct experiments on Application instances from the SAT Competition (SC) 2016, including all formulas that are unsatisfiable (UNSAT), and all those cannot be solved by any solver in the competition (UNKNOWN). These UNSAT and UNKNOWN formulas are a good resource of structured instances for testing MaxSAT algorithms. SAT solvers terminate once they identify a conflict, but MaxSAT solvers can provide useful information. For example, some of these formulas have only a few (even 1) unsatisfied clauses under the optimal solution, and such formulas can be modified to become satisfiable by removing the unsatisfied clauses under the solution.

Experiment setup: All experiments in this work are carried out on a cluster of workstations equipped with Intel Xeon E7-8830 2.13 GHz CPU, 24MB L3 cache and 1.0TB RAM under the operating system CentOS (version: 7.0.1406). Each solver is executed with the *runsolver* software [Roussel, 2011] for each instance with a cutoff time of 300 seconds (as in the incomplete track of MSEs), and the cost of the best found solution and the time for finding it (the last 'o' line) is recorded. For each solver on each benchmark, we report the number of instances where the solver finds the best solution among all competing solvers in the experiment, denoted by '#win.', and the averaged time of doing so on such winning instances, denoted by 'time' (in CPU seconds).

5.2 Experiment Results on DeciLS

We summarize the experiment results on all unweighted benchmarks in Table 1. DeciLS is overall the best solver on these benchmarks and is particularly effective on the industrial/application benchmarks.

DeciLS is the best solver for all the benchmarks except the crafted benchmark, where it is slightly worse then CCLS. This demonstrates the robustness of DeciLS. Comparatively,

Table 1: Experimental Results of DeciLS and state of the art solvers on all unweighted MaxSAT benchmarks.

Benchmark	#inst.	*DeciLS*		*CCLS*		*WPM3-2015-in*		*MiniWalk*		*dsat-wpm3-in-ms*		*VBS*	
		#win.	time	#win.	time	#win.	time	#win.	time	#win.	time	#win.	time
MSE2016_Random	454	**454**	**0.04**	**454**	0.73	0	0	6	51.88	403	10.53	454	0.03
MSE2016_Crafted	402	397	0.02	**402**	1.93	10	13.18	1	1.99	394	7.35	402	1.88
MSE2016_Industrial	55	**44**	55.46	0	0	42	38.34	1	14.77	43	44.90	55	39.14
SC_App_Unknown	103	**80**	102.81	8	122.93	1	3.78	39	62.25	1	15.55	103	101.63
SC_App_Unsat	109	**99**	92.32	22	59.53	14	27.32	34	44.04	12	18.28	109	77.86

Table 2: Comparison of *DeciLS* and its alternative versions as well as *DistUP* on all unweighted MaxSAT benchmarks.

Benchmark	#inst.	*DeciLS*		*DeciLS_onlyLS*		*DeciLS_onlyDeci*		*DistUP*		*VBS*	
		#win.	time	#win.	time	#win.	time	#win.	time	#win.	time
MSE2016_Random	454	**454**	**0.04**	**454**	0.06	2	94.64	426	7.49	454	0.02
MSE2016_Crafted	402	398	0.72	**400**	1.55	6	<0.01	396	1.23	402	1.60
MSE2016_Industrial	55	**54**	63.66	0	<0.01	15	26.69	7	2.31	55	57.20
SC_App_Unknown	103	**76**	107.11	38	29.92	10	111.13	12	80.06	103	101.02
SC_App_Unsat	109	**95**	91.32	40	49.44	17	69.31	12	147.04	109	85.93

Table 3: Experimental Results of *DeciDist*, *Dist* and *DistUP* on the MSE2016 PMS benchmarks.

Benchmark	#inst.	*DeciDist*		*Dist*		*DistUP*		*VBS*	
		#win.	time	#win.	time	#win.	time	#win.	time
MSE2016_PMS_Random	210	180	17.75	**208**	1.51	**208**	**0.71**	209	0.33
MSE2016_PMS_Crafted	678	**494**	39.16	357	7.82	391	5.69	566	33.67
MSE2016_PMS_Industrial	601	**398**	84.91	225	57.31	260	43.47	504	79.83

Table 4: Experimental Results of *DeciDist* and *WPM3-2015-in* on the MSE2016 PMS benchmarks.

Benchmark	#inst.	*DeciDist*		*WPM3-2015-in*		*VBS*	
		#win.	time	#win.	time	#win.	time
PMS_Random	210	**206**	22.05	39	74.38	209	21.46
PMS_Crafted	678	417	16.72	**608**	24.79	678	22.84
PMS_Industrial	601	221	50.44	**553**	26.01	601	29.30

other solvers including the portfolio solver have much worse performance than DeciLS on at least three benchmarks.

DeciLS is particularly effective and outperforms its competitors on industrial/application benchmarks. This is remarkable, as industrial benchmarks have been dominated by SAT-based solvers for years. This is the first time a approach that does not call a SAT solver has better performance on such instances. Besides, DeciLS performs very well on also random and crafted instances at the same time, where SAT-based solvers are much worse.

5.3 Experiment Analysis on DeciLS

DeciLS combines a decimation algorithm and a local search algorithm CCAnr. In order to show the effectiveness of the cooperation of these two algorithms in the framework, we test two alternatives, denoted as DeciLS_onlyDeci and DeciLS_onlyLS respectively. DeciLS_onlyDeci calls the decimation algorithms iteratively, and records the best found solution. DeciLS_onlyLS is indeed the local search solver without modifications made in DeciLS. Since DistUP [Cai *et al.*, 2016] uses UP in the initialization and then performs local search, we also compare DeciLS with DistUP.

Seen from Table 2, DeciLS significantly outperforms the alternatives and DistUP on inudstrial/application benchmarks, while it has almost the same performance with the local search solver on random and crafted benchmarks. The results indicate that the cooperation is critical in our approach, and that the approach is particularly effective for solving industrial instances.

We are also interested in the number of rounds of DeciLS to get the best found solution for each instance. We calculate the averaged number for each MaxSAT benchmark, which are as follows: 1.87 for Random, 7.89 for Crafted, 106.56 for Industrial, and 542.35 for App_Unknown, 384.77 for App_Unsat.

6 A New Incomplete Approach to (W)PMS

In this section, we extend our approach to solving (Weighted) Partial MaxSAT. The extension method for WPMS is simply the same one for PMS, and thus we only present the extension to PMS. To extend the approach and make it effective for PMS, some modifications have been carried out as follows:

Firstly, we extend the UP-based decimation algorithm (Algorithm 1) to PMS. When there exist both hard and soft unit clauses, our decimation algorithm for PMS prefers to pick a hard unit clause to perform unit propagation. Similarly, if conflicts are detected, the one that involves hard clauses is addressed first by assigning the corresponding variable according to the heuristic.

Secondly, we must deal with the situation that the previous local search process fails to find a feasible solution. In

Table 5: Experimental Results of *DeciCCEHC*, *CCEHC* and *CCEHC+UP* on the MSE2016 WPMS benchmarks.

Benchmark	#inst.	*DeciCCEHC*		*CCEHC*		*CCEHC+UP*		*VBS*	
		#win.	time	#win.	time	#win.	time	#win.	time
MSE2016_WPMS_Random	502	**501**	**1.12**	**501**	1.32	**501**	1.24	501	0.72
MSE2016_WPMS_Crafted	331	**262**	40.66	238	28.56	248	28.14	320	51.35
MSE2016_WPMS_Industrial	630	**319**	117.67	140	103.74	224	113.95	553	129.43

Table 6: Experimental Results of *DeciCCEHC* and *WPM3-2015-in* on the MSE2016 WPMS benchmarks.

Benchmark	#inst.	*DeciCCEHC*		*WPM3-2015-in*		*VBS*	
		#win.	time	#win.	time	#win.	time
WPMS_Random	502	**501**	1.12	12	107.89	501	1.12
WPMS_Crafted	331	**232**	42.21	180	29.21	331	41.04
WPMS_Industrial	630	138	72.72	**527**	38.77	630	46.82

this case, when UP cannot be performed, the decimation algorithm cannot consult the best found solution by previous local search, and we simply assign the variable randomly.

6.1 Experiments on PMS

We choose Dist [Cai *et al.*, 2016] as the local search solver for PMS in our approach. Dist has been ranked 1st many times in PMS categories of recent MSEs. The resulting hybrid solver is thus named DeciDist. The DeciDist is implemented on the codes of Dist (version 2016), and adopts the default parameter setting there. $MaxNonImprove$ is set to 10000000 for local search in DeciDist. We evaluate DeciDist on all PMS benchmarks in the MSE 2016.

First, we compare DeciDist with two local search PMS solvers namely Dist and DistUP. DistUP improves Dist slightly on PMS industrial instances. The results are reported in Table 3. Compared to the local search solvers, DeciDist has worse performance on PMS random instances. However, DeciDist shows a dramatic improvement over both local search solvers (noting that DistUP uses UP in its initialization) on PMS crafted and industrial benchmarks.

We also compare DeciDist with WPM3-2015-in, which placed 2nd and is the best non-portfolio solver in PMS crafted and industrial categories of the incomplete track in the MSE 2016. We summarize the comparison results in Table 4. Unsurprisingly, the performance of the SAT-based solver WPM3-2015-in is much worse on random instances. On the other hand, WPM3-2015-in has better performance than DeciDist on PMS crafted and industrial benchmarks. Nevertheless, DeciDist still finds better solutions than WPM3-2015-in for 70 PMS crafted instances and 48 PMS industrial instances. This indicates that Decidist is complementary to WPM3-2015-in on these two benchmarks.

The averaged rounds of DeciDist to get the best found solution for PMS benchmarks are: 1.43 for PMS Random, 6.42 for PMS Crafted and 11.83 for PMS Industrial.

6.2 Experiments on WPMS

We use this approach to improve a state of the art local search solver for WPMS called CCEHC [Luo *et al.*, 2017], resulting in a solver named DeciCCEHC. We first compare DeciCCEHC with CCEHC and CCEHC+UP [Luo *et al.*, 2017], where the later is an improved version of the former by using a UP based initialization as in DistUP. We then compare DeciCCEHC with the SAT-based solver WPM3-2015-in. The experiments are carried out on all WPMS benchmarks in the MSE 2016.

The experiment results for WPMS benchmarks are presented in Tables 5 and 6. Our solver DeciCCEHC outperforms both local search solvers on all WPMS benchmarks, and outperforms WPM3-2015-in on WPMS crafted benchmark, although it is worse on industrial benchmark.

The averaged rounds of DeciCCEHC to get the best found solution for WPMS benchmarks are: 1 for WPMS Random, 1.26 for WPMS Crafted and 113.94 for WPMS Industrial.

7 Conclusions

In this work, we proposed a new incomplete approach to MaxSAT and Partial MaxSAT. The approach is based on a new framework that interleaves between a decimation algorithm and a local search algorithm. We proposed a novel decimation algorithm for MaxSAT which is based on unit propagation, and used it in our framework. The resulting MaxSAT solver named DeciLS showed state of the art performance on random, crafted and industrial benchmarks, and outperforms state of the art SAT-based solvers on industrial benchmarks. We then extended the approach to solving (Weighted) Partial MaxSAT, and the experiments showed that this approach significantly improves the performance of local search on (W)PMS crafted and industrial benchmarks and is complementary to SAT-based solvers.

We believe this new approach is promising for MaxSAT solving and deserves further research. We would like to further improve the approach for (W)PMS problems.

Acknowledgements

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

References

[Altarelli *et al.*, 2009] Fabrizio Altarelli, Rémi Monasson, Guilhem Semerjian, and Francesco Zamponi. Connections to statistical physics. In *Handbook of Satisfiability*, pages 569–611. 2009.

[Ansótegui *et al.*, 2015] Carlos Ansótegui, Frédéric Didier, and Joel Gabàs. Exploiting the structure of unsatisfiable cores in maxsat. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 283–289, 2015.

[Ansótegui *et al.*, 2016] Carlos Ansótegui, Joel Gabàs, and Jordi Levy. Exploiting subproblem optimization in SAT-based MaxSAT algorithms. *J. Heuristics*, 22(1):1–53, 2016.

[Ansótegui *et al.*, 2017] Carlos Ansótegui, Josep Pon, Meinof Sellmann, and Kevin Tierney. Reactive dialectic search portfolios for MaxSAT. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), San Francisco, California USA*, page to appear, 2017.

[Braunstein *et al.*, 2005] A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Struct. Algorithms*, 27(2):201–226, 2005.

[Cai *et al.*, 2015] Shaowei Cai, Chuan Luo, and Kaile Su. CCAnr: A configuration checking based local search solver for non-random satisfiability. In *Proceedings of18th International Conference on Theory and Applications of Satisfiability Testing, SAT 2015, Austin, TX, USA, September 24-27, 2015*, pages 1–8, 2015.

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

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

[Chao and Franco, 1986] Ming-Te Chao and John V. Franco. Probabilistic analysis of two heuristics for the 3-satisfiability problem. *SIAM J. Comput.*, 15(4):1106–1118, 1986.

[Chao and Franco, 1990] Ming-Te Chao and John V. Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k satisfiability problem. *Inf. Sci.*, 51(3):289–314, 1990.

[Chen *et al.*, 2010] Yibin Chen, Sean Safarpour, João Marques-Silva, and Andreas G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(11):1804–1817, 2010.

[Chieu and Lee, 2009] Hai Leong Chieu and Wee Sun Lee. Relaxed survey propagation for the weighted maximum satisfiability problem. *J. Artif. Intell. Res. (JAIR)*, 36:229–266, 2009.

[Darwiche and Pipatsrisawat, 2009] Adnan Darwiche and Knot Pipatsrisawat. Complete algorithms. In *Handbook of Satisfiability*, pages 99–130. IOS Press, 2009.

[Eén and Biere, 2005] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proc. of SAT-05*, pages 61–75, 2005.

[Hirsch and Kojevnikov, 2005] Edward A. Hirsch and Arist Kojevnikov. Unitwalk: A new SAT solver that uses local search guided by unit clause elimination. *Ann. Math. Artif. Intell.*, 43(1):91–111, 2005.

[Hutter *et al.*, 2011] Frank Hutter, Holger H. Hoos, and 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*, pages 507–523, 2011.

[Janota *et al.*, 2012] Mikolás Janota, Inês Lynce, Vasco M. Manquinho, and João Marques-Silva. PackUp: Tools for package upgradability solving. *JSAT*, 8(1/2):89–94, 2012.

[Kroc *et al.*, 2009] Lukas Kroc, Ashish Sabharwal, Carla P. Gomes, and Bart Selman. Integrating systematic and local search paradigms: A new strategy for MaxSAT. In *Proc. of IJCAI-09*, pages 544–551, 2009.

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

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

[Paturi *et al.*, 2005] Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for *k*-SAT. *J. ACM*, 52(3):337–364, 2005.

[Roussel, 2011] Olivier Roussel. Controlling a solver execution with the runsolver tool. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(4):139–144, 2011.

[Safarpour *et al.*, 2007] Sean Safarpour, Hratch Mangassarian, Andreas G. Veneris, Mark H. Liffiton, and Karem A. Sakallah. Improved design debugging using maximum satisfiability. In *Proc. of FMCAD-07*, pages 13–19, 2007.

[Selman *et al.*, 1994] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proc. of AAAI-94*, pages 337–343, 1994.

[Xu *et al.*, 2003] Hui Xu, Rob A. Rutenbar, and Karem A. Sakallah. sub-SAT: a formulation for relaxed boolean satisfiability with applications in routing. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(6):814–820, 2003.