Solving Set Cover and Dominating Set via Maximum Satisfiability

Zhendong Lei, Shaowei Cai[*]
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
School of Computer and Control Engineering, University of Chinese Academy of Sciences
leizd@ios.ac.cn, shaoweicai.cs@gmail.com

Abstract

The Set Covering Problem (SCP) and Dominating Set Problem (DSP) are NP-hard and have many real world applications. SCP and DSP can be encoded into Maximum Satisfiability (MaxSAT) naturally and the resulting instances share a special structure. In this paper, we develop an efficient local search solver for MaxSAT instances of this kind. Our algorithm contains three phrase: construction, local search and recovery. In construction phrase, we simplify the instance by three reduction rules and construct an initial solution by a greedy heuristic. The initial solution is improved during the local search phrase, which exploits the feature of such instances in the scoring function and the variable selection heuristic. Finally, the corresponding solution of original instance is recovered in the recovery phrase. Experiment results on a broad range of large scale instances of SCP and DSP show that our algorithm significantly outperforms state of the art solvers for SCP, DSP and MaxSAT.

Introduction

Many combinatorial optimization problems that arise in the real world are difficult to solve partly because they present computational challenges. One of the most effective approach to solve such problems is to first model them in a mathematical or logical language, and then solve them by applying a suitable algorithm. This paper is concerned with developing a practical algorithm to solve two optimization problems modelled in a particular logical language, Maximum Satisfiability (MaxSAT) .

Given a propositional formula expressed in the Conjunctive Normal Form (CNF), Satisfiability (SAT) concerns about satisfying all clauses. MaxSAT is an optimization version of SAT, and its most general form contains both hard clauses and weighted soft clauses. Such a MaxSAT problem is referred to as weighted Partial MaxSAT (WPMS), where the goal is to find an assignment that satisfies all the hard clauses and maximize the total weight of satisfied soft clauses.

There is a remarkable number of algorithms for (W)PMS including complete solvers (Davies and Bacchus 2011; Ansótegui, Bonet, and Levy 2013; Narodytska and Bacchus 2014; Martins, Manquinho, and Lynce 2014; Ansótegui and Gabàs 2017) and incomplete solvers (Cai et al. 2014; 2016; Luo et al. 2017; Lei and Cai 2018). Due to the success of these works, there has been much interest in solving combinatorial optimization problems by this kind of logical language (Demirovic and Musliu 2017; Jiang et al. 2018; Demirovic, Musliu, and Winter 2019).

In real world, WPMS solvers are usually used for solving problems from some specific domain. Each specific problem has its own characteristics which are important for designing algorithms. However, previous WPMS solvers are usually developed for general purpose and do not exploit the feature of the instances from specific domains. In our opinion, exploiting these features in WPMS solving would lead to more effective solvers for WPMS from specific domains.

In this paper, we propose a WPMS solver to solve two closely related NP hard combinatorial optimization problems of importance, namely Set Cover Problem (SCP) and Dominating Set Problem (DSP). SCP has many applications, from crew scheduling in railway and mass-transit companies to job assignment in manufacturing and service location (Caprara et al. 1997; Bautista and Pereira 2006). DSP also finds valuable applications in many fields (Hedetniemi et al. 2003; Aoun et al. 2006; Chalupa 2018). For example, Shen and Li (Shen and Li 2010) solve the multi-document problem by encoding this problem to DSP.

Many works have been done to solve SCP and DSP over the years. For SCP, there are complete algorithms (Caprara, Toth, and Fischetti 2000), genetic algorithm (Beasley and Chu 1996), simulated annealing (Jacobs and Brusco 1995) and local search algorithm (Gao, Weise, and Li 2014). For DSP, there are ant colony optimization (ACO) (Jovanovic et al. 2010), swarm intelligence algorithm (Nitash and Singh 2014) and local search algorithm (Wang et al. 2018).

In this work, we formulate SCP and DSP into WPMS and design a specific local search algorithm (denoted as DomSAT) to solve them. The new algorithm contains three phases: construction, local search and recovery. The main contributions of this paper are as follows:

First, based on the features of such WPMS instances, we propose three reduction rules to simplify the instances.

[*]Corresponding author

Then, we construct a feasible solution greedily for the simplified instance, which serves as the initial solution for the local search phrase. The local search algorithm is mainly based on a variable selection heuristic, which distinguishes four different situations during the search, and employs different variable selection rules under each situation. Finally, by extending the assignment returned by local search phase with data structure of the reduction rules, we obtain the corresponding solution of the original intance.

We carry out experiments to compare our algorithm DomSAT with state of art solvers for SCP, DSP and WPMS on a broad range of benchmarks. According to the experiments, DomSAT performs significantly better than all the competitors. We also perform experimental analysis and additional investigations to show the effectiveness of the proposed heuristics.

The reminder of this paper is organized as follows. The next section introduces preliminary knowledge. Section 3 presents the encodings from SCP and DSP to WPMS, and the reduction rules. Section 4 present an overview of our method. Then, the construction procedure and recovery procedure are introduced in Section 5. The main local search procedure is presented in Section 6. Experimental results are presented in Section 7. Finally, we conclude the paper.

Preliminaries

SCP: Given an universal set X and a set Y which contains subsets of X with $\cup_{\forall y \in Y} = X$, each element in Y is associated with a weight $w(y)$, the goal is to find a set $F \subseteq Y$ of the smallest total weight but still contains all elements in X, that is, $\cup_{\forall y \in F} = X$. We use $U = (X, Y, W)$ to denote a SCP instance.

DSP: An undirected graph $G = (V, E)$ consists of a vertex set V and an edge set $E \subseteq V \times V$ in which each edge is a 2-element subset of V. A vertex weighted undirected graph is an undirected graph in which each vertex $v \in V$ is associated with an integer number $w(v)$. We use $G = (V, E, W)$ to denote a vertex weighted graph. Given a vertex weighted undirected graph $G = (V, E, W)$, the Dominating Set Problem (DSP) is to find a subset D with the smallest total weight of its vertices such that every vertex either belongs to D or is adjacent to at least one vertex in D.

WPMS: Given a set of Boolean variables $\{v_1, v_2, \ldots, v_n\}$, a literal is either a variables v_i or its negation $\neg v_i$. A clause is a disjunction of literals which can be expressed as a set of literals. A clause is satisfied if it has at least one true literal, and falsified otherwise. An empty clause $\square$ is falsified. A conjunctive normal form (CNF) formula F is a conjunction of clauses which can be expressed as a set of clauses. Two variables are neighbors iff they occur in at least one clause. Let $N(v) = \{u | u \in V(F)$ and u occurs in at least one clause with $v\}$, which is the set of all neighboring variables of variable v.

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

A complete assignment for (W)PMS is a mapping that assigns to each variable either 0 or 1. For an assignment α, we use $\alpha[v]$ to denote the assignment of variable v under α. For a (W)PMS instance F, we say an assignment α is feasible iff it satisfies all hard clauses in F. The $cost$ of a feasible assignment α is defined to be the number (resp. the sum of weights) of falsified soft clauses under α for PMS (resp. WPMS). In order to generalize this concept to infeasible assignments, we use $soft\ cost$ ($cost_s$) to refer to this measure for infeasible assignments.

A probabilistic sampling strategy named best from Multiple Selections(BMS) (Cai 2015) chooses t (t is an integer parameter) candidates randomly with replacement from the set of all candidates and returns the best one we want. In our algorithm, t is set to 100 for small instances (< 10000 variables) and 60 for large instances (> 10000 variables).

Encodings and Reduction Rules

SCP and DSP can be encoded into WPMS instances naturally. Based on the observations on such WPMS instances, we propose several reduction rules to simplify the instances.

Encodings

For convenience, we present our encodings with the set form of clause, i.e., a clause is expressed as a set of literals. In this work, we consider SCP and DSP on weighted graphs, and the unweighted versions can be viewed as a special case of the weighted versions where each weight is equally one.

Given a SCP instance $U = (X, Y, W)$, we encode it into a WPMS instance as follows:

- Variables: For each element (a subset) in Y, the WPMS instance has a boolean variable v_y that represents whether or not the subset y is selected in the solution.
- Hard clauses: For each element $x \in X$, generate a hard clause $\{v_y | y \in Y, x \in y\}$, to ensure that each element in X is covered by at least one subset in Y.
- Soft clauses: For each element $y \in Y$, generate a soft clause $\{\neg v_y\}$ and its weight is equal to $w(y)$.

Given a weighted graph $G = (V, E, W)$, we encode a DSP instance into a WPMS instance as follows:

- Variables: For each vertex in $i \in V$, the WPMS instance has a boolean variable v_i that that represents whether or not the vertex i is selected in the solution.
- Hard clauses: For each vertex $i \in V$, generate a hard clause $\{v_j | (i, j) \in E\} \cup \{v_i\}$, to ensure that each vertex in V is dominated.
- Soft clauses: For each vertex $i \in V$, generate a soft clause $\{\neg v_i\}$ and its weight is equal to $w(i)$;

The resulting WPMS instances encoded from SCP and DSP have two significant features:

- (a) Hard clauses only contain positive literals.
- (b) For each variable v, there is exactly one soft clause, which is $\{\neg v\}$.

Reduction Rules

We propose several reduction rules to simplify the instances with the two characteristics. A key concept underlying our reduction rules is the "dominating" relation between variables, which is formally defined below.

Definition 1 (**dominating relations**). *For any (W)PMS instance with the two features (a) and (b) mentioned above,*

- *a variable v' dominates another variable v, denoted as $v' \triangleright v$, if for any hard clause c, if $v \in c$ then $v' \in c$.*
- *a variable v' strong dominates another variable v, denoted as $v' \succ v$, if $v' \triangleright v$ and $w(\{\neg v'\}) \leq w(\{\neg v\})$.*

With the above definition, we present our reduction rules below.

Hard Unit Clause Rule: If there is a hard unit clause $c = \{v\}$, then variable v is assigned to 1 in order to satisfy the unit clause.

The Hard Unit Clause Rule is trivially sound and indeed applies to any WPMS instance. When it is applied to a instances with features (a) and (b), with a hard unit clause $c = \{v\}$, the WPMS instance is simplified by deleting all clauses containing the literal v (as they are satisfied) and the soft clause $\{\neg v\}$ becomes an empty clause.

Strong Dominating Rule: For a variable v, if $\exists v'(v' \succ v)$, then v is assigned to 0.

When Strong Dominating Rule is applied to a WPMS instances with features (a) and (b), the WPMS instance is simplified by deleting the soft clause $\{\neg v\}$ (as it is satisfied), and removing the literal v from all hard clauses containing literal v.

Binary Weak Dominating Rule: For a variable v, if there is a binary hard clause $v \vee v'$ and $v' \triangleright v$, then rewrite v as $\neg v'$.

When Binary Weak Dominating Rule is applied to a WPMS instances with features (a) and (b), the WPMS instance is simplified as follows: first, delete all hard clauses that contain $v \vee v'$ (which is $\neg v' \vee v'$ after rewriting, and thus is a tautology); then, soft clauses $(\{\neg v\}, w(\{\neg v\}))$ and $(\{\neg v'\}, w(\{\neg v'\}))$ beome $(\{\Box\}, w(\{\neg v\}))$ and $(\{\neg v'\}, w(\{\neg v'\}) - w(\{\neg v\}))$.

Let us denote the original WPMS instance as F and the simplified formula as F'. It is easy to see that, for both dominating rules, every feasible solution for F' corresponds to a feasible solution in F with the same cost. Nevertheless, the proof of the optimal-reserving property (i.e., F and F' have the same optimal cost) of these two rules is not trivial, and is outlined below.

Proof: (1) Optimal-reserving of Strong Dominating rule: Suppose F has an optimal solution α with $\alpha[v] = 1$ where v is the variable fixed to 0 by the rule. According to the rule, we know that there exists a variable v' such that $v' \triangleright v$ and $w(\{\neg v'\}) \leq w(\{\neg v\})$. We can construct a solution α' which is modified from α by flipping the value of v to 0 and setting $\alpha'[v'] = 1$. We will show that $cost(\alpha') \leq cost(\alpha)$.

Firstly, α' satisfies all hard clauses. It is sufficient to consider the hard clauses containing v. Such clauses also contain v' (as $v' \triangleright v$), and by setting $\alpha'[v'] = 1$, they are satisfied. As for the cost, if $\alpha[v'] = 0$ then $cost(\alpha') = cost(\alpha) - w(\{\neg v\}) + w(\{\neg v'\}) \leq cost(\alpha)$; if $\alpha[v'] = 1$, then $cost(\alpha') = cost(\alpha) - w(\{\neg v\}) < cost(\alpha)$. Hence, $cost(\alpha') \leq cost(\alpha)$ always holds. Therefore, it is safe to fix v to 0 without changing the optimal cost of the WPMS instance.

(2) Optimal-reserving of Binary Weak Dominating rule: If there is a binary hard clause $v \vee v'$ and $v' \triangleright v$, then $\alpha[v] = 1 - \alpha[v']$ in any optimal solution α. We prove this by contradiction. Suppose there is an optimal solution α with $\alpha[v] = \alpha[v']$. Since α satisfies all hard clause, we conclude that $\alpha[v] = \alpha[v'] = 1$, as $\alpha[v] = \alpha[v'] = 0$ would make the hard clause $x \vee y$ falsified. Now, if we flip v to 0, it would lead to a feasible solution with lower cost, which contradicts with our assumption. $\Box$

To illustrate the reduction rules, we provide a small example below.

- Originally, the WPMS instance F has 8 hard clauses as $h_1 := \{v_1, v_2\}$, $h_2 := \{v_1, v_2, v_3, v_4\}$, $h_3 := \{v_2, v_3, v_5\}$, $h_4 := \{v_2, v_4, v_5\}$, $h_5 := \{v_3, v_4, v_5\}$, $h_6 := \{v_5, v_6, v_7\}$, $h_7 := \{v_6, v_7, v_8\}$ $h_8 := \{v_7, v_8\}$ and 8 soft clauses with their weights as $(\{\neg v_1\}, 2)$, $(\{\neg v_2\}, 3)$, $(\{\neg v_3\}, 2)$, $(\{\neg v_4\}, 2)$, $(\{\neg v_5\}, 7)$, $(\{\neg v_6\}, 5)$, $(\{\neg v_7\}, 3)$, $(\{\neg v_8\}, 4)$.
- By Strong Dominating Rule, v_8 is assigned to 0, then $\{\neg v_8\}$ is deleted and h_7, h_8 become $\{v_6, v_7\}$ and $\{v_7\}$ respectively.
- By Hard Unit Clause Rule, v_7 is assigned to 1, then h_6, h_7, h_8 are deleted and $(\{\neg v_7\}, 3)$ becomes $(\Box, 3)$.
- By Weak Dominating Rule, v_1 is rewrite to $\neg v_2$, then h_1, h_2 are deleted, and $(\{\neg v_1\}, 2)$ and $(\{\neg v_2\}, 3)$ become $(\{\Box\}, 2)$ and $(\{\neg v_2\}, 1)$ respectively.
- Finally, the simplified WPMS instances F' has 3 hard clauses as $h_3 := \{v_2, v_3, v_5\}$, $h_4 := \{v_2, v_4, v_5\}$, $h_5 := \{v_3, v_4, v_5\}$ and 7 soft clauses $(\{\Box\}, 2)$, $(\{\neg v_2\}, 1)$, $(\{\neg v_3\}, 2)$, $(\{\neg v_4\}, 2)$, $(\{\neg v_5\}, 7)$, $(\{\neg v_6\}, 5)$, $(\{\Box\}, 3)$

Note that, most instances do not have hard unit clause. However, the applications of other rules may lead to hard unit clauses, as illustrated in the above example.

Algorithm 1: DomSAT Algorithm

```
  Input: PMS instance $F$
  Output: A feasible assignment $\alpha$ of $F$ and its cost
1 begin
2     ($\alpha_0$, $F'$, $RelVar$) := Construction(F);
3     $\alpha$ := LocalSearch($F'$, $\alpha_0$, cutoff_time);
4     $\alpha$ := Recovery($\alpha$, $RelVar$);
5     return ($\alpha$, $cost(\alpha)$);
```

Top-level Framework and Scoring Functions

In this section, we give a brief overview of our new algorithm DomSAT. As shown in Algorithm 1, DomSAT contains three phases: construction, local search and recovery.

In the construction phase, the original WPMS instance F is simplified by the reduction rules, resulting in a simplified instance F'. Then, a feasible assignment α_0 for F' is constructed by a greedy heuristic. Note that, we also record the data structure of the reduction rules, which will be used in the final phase to obtain the final solution to the original instance.

After that, a local search algorithm is called to solve the simplified instance F', with α_0 as the initial solution. This leads to an improved solution α.

Finally, it is easy to obtain a feasible solution for the original instance F by extending the assignment α returned by local search, with the help of data structure on the execution of the reduction rules.

In our algorithm, three scoring functions are used for picking the variable to flip. The first two are hard score and soft score (Cai et al. 2014).

Definition 2 (**hard score**). *The hard score of a variable x, denoted by $hscore(x)$, is the increase of the total weight of satisfied hard clauses by flipping x.*

Definition 3 (**soft score**). *The soft score of a variable x, denoted by $sscore(x)$, is the increase of the total weight of satisfied soft clauses by flipping x.*

The features of WPMS instances from SCP and DSP lead to some important properties on hard score and soft score.

- If the current value of variable v is 1, then $hscore(v) \leq 0$ and $sscore(v) > 0$;
- If the current value of variable v is 0, then $hscore(v) \geq 0$ and $sscore(v) < 0$;
- As each variable v appears in only one soft clause $\{\neg v\}$, the absolute value of $sscore(v)$ is always equal to $w(\{\neg v\})$.

Besides these two functions, we design a special scoring function, based on the features of the WPMS instances from SCP and DSP.

Definition 4 (**score**). *The score of a variable x is denoted by $score(x)$ s.t. $score(x) = hscore(x)/|sscore(x)|$*

Hard clause weighting scheme is an effective technique in local search algorithms for WPMS. In this work, we also employ a hard clause weighting scheme. With this scheme, the clause weights in the score functions are defined as follows. For soft clauses, we use the original weight as its weight which will not be modified during the search. For hard clauses, the weights are updated according to the weighting scheme. Specifically, each hard clause c is associated with an integer number as its weight, which is initialized to 1. And the weight of each falsified hard clause is increased by one at the end of each iteration of local search.

For WPMS instances from SCP and DSP, flipping a variable x to satisfy more hard clauses always comes with an increment on the soft cost, which is equal to $|sscore(x)|$. This $score$ function considers both merits and measures the ratio.

Construction and Recovery Procedure

Before we present the details of the Construction and Recovery procedures, we first introduce the key data structure.

- $RelVar[x] = y$ indicates that the value of x refers to the value of y, specifically that the value of x is the opposite of y.
- $RelVar[x] = -1$ indicates that the value of x does not refer to any variable.

Algorithm 2: The Construction Procedure

```
Input: PMS instance $F$
Output: A feasible assignment $\alpha$, simplified instance
        $F$, $RelVar$
1  begin
2      For each variable x, $RelVar[x] := -1$,
         $\alpha[x] := -1$;
3      while any reduction rules are satisfied do
4          if Hard Unit Clause Rule is satisfied then
5              Apply Unit Clause Rule;
6          else if Strong Dominating Rule is satisfied then
7              Apply Strong Dominating Rule;
8          else if Binary Weak Dominating Rule is satisfied
             then
9              Apply Binary Weak Dominating Rule;
10             Update $RelVar$ accordingly;
11     For each variable x with $\alpha[x] = -1$, $\alpha[x] := 0$;
12     while There exist falsified hard clause do
13         $c$ := a randomly falsified hard clause;
14         $v$ := a variable in $c$ with the biggest score;
15         $\alpha := \alpha$ with $v$ flipped;
16     return $(\alpha, F, RelVar)$
```

The construction procedure consists two parts: reduction (line 2-11) and greedy construction (line 12-16). In the reduction part, if any reduction rules are satisfied, the algorithm simplifies the instance accordingly. In the construction procedure, during each step, $ConstructionAlg$ picks a falsified hard clause c randomly. Then, the variable with the biggest score in c is picked and flipped untill there is no falsified hard clauses. Note that, we can always find a feasible solution because hard clauses only contains positive literals. The construction procedure play an important role in our new algorithm.

The Recovery phase is very simple: it scans all variables in the original WPMS instance, and for each varaiable v such that $RelVar[v] \neq -1$, then v is assigned to 1-$\alpha[RelVar[v]]$. This is for assigning the variables that have been rewritten by the Binary Weak Dominating Rule.

Local Search Procedure

Before we present the details of the Local Search Procedure, we first introduce the Configuration Checking strategy and variable selection heuristic used in our algorithm.

Configuration Checking Strategy

Firstly proposed in (Cai and Su 2011), configuration checking (CC) is a strategy aiming to prevent cycles in search, by forbidding flipping a variable x if none of its neighboring variables has changed the value since x's last flip. We propose a CC strategy that is tailored for WPMS instances from SCP and DSP. The checking mechanism of our CC strategy is one-side, only working on variables with $\alpha[v] = 0$.

To implement the CC strategy, we employ an array $confChange$, where $confChange[v] = 1$ means at least one variables in $N(v)$ has been flipped since v's last flip; and $confChange[v] = 0$ on the contrary. The CC strategy can be described as follows:

Updating rules: For each variable v, $ConfChange[v]$ is initialized to 1. When a variable v is flipped from 1 to 0, $ConfChange[v]$ is reset to 0; and whenever a variable v is flipped, $ConfChange[u]$ is set to 1 for variables $u \in N(v)$.

Using rules: While choosing a variable with $\alpha[v] = 0$, our new configuration checking strategy forbids any variable v to be flipped if its configuration has not been changed, i.e., $ConfChange[v] = 0$.

For a variable v with $\alpha[v] = 1$, flipping such a variable will decrease the total cost of current solution (because $sscore(v) > 0$), so we do not apply CC constrains in this situation.

Variable Selection Heuristic

Our variable selection heuristic depends on the existence of three particular types of variables, which are defined below.

Definition 5. *For a variable x, x is decreasing iff $hscore(x) = 0$ and $sscore(x) > 0$, and is hard-decreasing iff $hscore(x) > 0$ and $ConfChange[v] = 1$, and is soft-decreasing iff $sscore(x) > 0$.*

We distinguish four situations during the search, according to the existence of the above types of variables. Based on this, we design a variable selection heuristic with four levels.

Descent Mode (first level): The current assignment α is feasible and there exist decreasing variables. Flipping decreasing variables would decrease the cost without breaking any hard clause, leading to a better feasible solution. So, a decreasing variable is picked according to BMS strategy w.r.t. highest $sscore$. This is the fastest way towards the goal of WPMS under this situation.

Exploitation Mode (second level): If α is not feasible, we aim to get a feasible assignment while at the same time try to keep soft cost as small as possible. Thus, we pick a variable from a random falsified hard clause c, which would at least satisfy c. Specifically, a hard-decreasing variable v in c is picked by BMS strategy w.r.t. highest $score$. If flipping v would make the soft cost lower than $cost^*$ (the best found cost), then v is flipped. Otherwise, the algorithm goes to the third level.

Aspiration Mode (third level): Note that the second level usually fails if $cost^*$ is close to optimal. To address this issue, we propose the aspiration mode. When the second level fails, we pick a soft-decreasing variable u by BMS strategy w.r.t. $score$. And if $score(v)+score(u) > 0$, which means flipping these two variables is generally good, both v and u are flipped.

Perturbation Mode (fourth level): When all the preceding modes are not applicable, which means the search gets "stuck", the algorithm switches to the perturbation mode. Two soft-decreasing variables v_1 and v_2 are picked by BMS strategy w.r.t. $score$, to explore the search space.

The Local Search Procedure

Algorithm 3: Local Search Procedure

```
Input: PMS instance $F$, an complete assignment $\alpha$,
       cutoff
Output: A feasible assignment $\alpha$
1  begin
2      $\alpha^* := \alpha$, $cost^* := cost(\alpha)$;
3      while elapsed time < cutoff do
4          if $\alpha$ is feasible & $cost(\alpha) < cost^*$ then
5              $\alpha^* := \alpha$, $cost^* := cost(\alpha)$;
6          if $\alpha$ is feasible then
7              if there exist decreasing variables then
8                  $v$ := the decreasing variable by BMS
                     w.r.t. highest $sscore$;
9                  $\alpha$ := $\alpha$ with $v$ flipped;
10             else
11                 perturbation();
12         else
13             $c$ := a random falsified hard clause;
14             $v$ := a hard-decreasing variable in $c$ is
                 picked by BMS w.r.t. highest $score$;
15             if $cost_s(\alpha) - sscore(v) < cost^*$ then
16                 $\alpha$ := $\alpha$ with v flipped;
17             else
18                 $u$ := a soft-decreasing variable is
                     picked by BMS w.r.t. highest $score$;
19                 if $score(v) + score(u) > 0$ then
20                     $\alpha$ := $\alpha$ with v and u flipped;
21                 else
22                     perturbation();
23         update confChange and the weights of hard
             clause;
24     return $\alpha^*$;
```

Based on the variable selection heuristic in the preceding subsection, we develop an efficient local search algorithm for solving the simplified instances. The local search procedure is outlined in Algorithm 3.

In the beginning, the best found solution α^* is initialized as the input solution α. After that, a loop (line 3-23) is executed to iteratively modify α until a given time limit is reached. During the search, whenever a better feasible solution is found, the best feasible solution α^* and $cost^*$ are updated (line 4-5). At each iteration, DomSAT chooses a variables and flips it, according to the variable selection heuristic, with all ties broken by age.

First, if the current solution α is feasible and there exist decreasing variables, DomSAT picks a decreasing variable

Table 1: Results on unweighted Set Cover Problem

Instance	*DomSAT*		*WCC*		*SATLike*		*Loandra*		*Open-wbo*	
	cmin	time	cmin	time	cmin	time	cmin	time	cmin	time
STS135	**103**	8.98	**103**	4.13	**103**	52.1	104	65.83	104	158.44
STS243	**198**	0.11	**198**	0.18	**198**	0.74	201	402.35	202	117.26
STS405	**335**	321.29	**335**	91.11	340	296.78	346	699.56	344	836.03
STS729	**617**	36.41	**617**	61.16	639	404.77	643	5.97	652	614.74

Table 2: Summary Results on random weighted SCP instances

Benchmark	#inst.	*DomSAT*		*WCC*		*SATLike*		*Loandra*		*Open-wbo*	
		#win	time	#win	time	#win	time	#win	time	#win	time
OR_small	45	**45**	0.01	**45**	0.02	13	2.96	34	58.44	2	0.26
OR_large	20	**20**	0.92	**20**	0.72	3	4.10	9	81.96	4	1.70

by BMS w.r.t. $sscore$, breaking ties by age (line 6-9, Descent Mode). If α is feasible and there is no decreasing variable, then the algorithm performs perturbation by flipping two soft-decreasing variables picked by BMS w.r.t. $sscore$.

If α is infeasible, DomSAT chooses a falsified hard clause c randomly. Then DomSAT picks a hard-decreasing variable v by BMS strategy w.r.t. $score$, and checks whether $cost(\alpha) - sscore(v) < cost^*$. If this is the case, then v is flipped (line 15-16, Exploitation Model); otherwise, a soft-decreasing variable u is picked by BMS strategy w.r.t. $score$, and if $score(v) + score(u) > 0$, DomSAT flips v and u simultaneously (line 18-20, Aspiration Model). Finally, if all above modes fail, a perturbation step is executed to explore other areas of search space. At the end of each iteration, it updates confChange accordingly and increases the weight of each falsified hard clause by 1.

In the end, when the loop terminates upon reaching the time limit, DomSAT reports the best feasible solution α^* that has been found.

Experimental Evaluation

We evaluate our DomSAT algorithm on a wide range of benchmarks including many massive real world instances. These testing benchmarks are described in detail below.

- The first benchmark (OR-Library) (Beasley 1990) is a collection of test data sets for a variety of Operations Research (OR) problems, which was originally described by J.E.Beasley. OR-Library instances are weighted SCP instances are divided into two sets OR_small and OR_large according to their size.
- The second benchmark (Rail)[1] contains real-world weighted SCP instances that arise from an application in Italian railways and have been contributed by Paolo Nobili.
- The third benchmark (STS) (Fulkerson, Nemhauser, and Trotter 1974) which is from Steiner triple systems, contains unweighted SCP instances.

[1]http://people.brunel.ac.uk/ mastjjb/jeb/orlib/files/

- Moreover, we consider the 65 massive real world graphs from (Wang et al. 2018) as DSP instances. To obtain the corresponding weighted DSP instances, we use the same method as in (Wang et al. 2018), i.e., for the ith vertex v_i, $w(v_i)$=(i mod 200)+1.

Our algorithm is compared against five state of art algorithms including SCP, DSP and WPMS solvers.

- WCC (Gao, Weise, and Li 2014) is the best SCP solver which can work on weighted and unweighted version. The algorithm was not given a name, but since the main ideas of the algorithm include constraint Weighting and Configuration Checking, we denote it as WCC in our experiment.
- FastMWDS (Wang et al. 2018) is the best DSP solvers for weighted DSP instances.
- SATLike (Lei and Cai 2018) is the best local search solver for WPMS and won two unweighted categories of incomplete track in MSE (MaxSAT Evaluation) 2018.
- Open-WBO (Martins, Manquinho, and Lynce 2014) won the weighted 60s timeout category of incomplete track in MSE 2018 and its improved version TT-Open-WBO-inc won two weighted categories of incomplete track in MSE 2019. We compare with the best version TT-Open-WBO-inc, and we denote it as Open-WBO for convenience.
- Loandra (MaxSAT Evaluation 2019) won two unweighted categories, and was ranked 2nd in two weighted categories of incomplete track in MSE 2019.

Experiment Preliminaries

DomSAT is implemented in C++ and compiled by g++ with -O3 option. Our experiments were conducted on a server using Intel Xeon E7-8850 v2 @2.30GHz, 2048GB RAM, running Ubuntu 16.04.5 Linux operation system. Each solver is executed 10 times on each instance with different seeds. The time limit of all algorithms is 1000 seconds on all benchmarks except OR instances. For OR instances, the time limit of all algorithm is 10 seconds. In each run, the solver prints successively the best solution it has found so far. For each instance, $cmin$ is the cost of the best feasible solution found.

Table 3: Results on industrial weighted SCP instances

Instance	*DomSAT*		*DomSAT-*		*WCC*		*SATLike*		*Loandra*		*Open-wbo*	
	cmin	time	cmin	time	cmin	time	cmin	time	cmin	time	cmin	time
rail1	**703**	718.02	712	19.76	732	884.17	N/A	N/A	1106	998.99	1123	5.06
rail2	**967**	721.80	979	19.85	1006	945.08	N/A	N/A	1433	990.10	1464	4.96
rail3	**1100**	770.77	1116	19.21	1148	906.89	N/A	N/A	1903	28.28	1734	6.07
rail4	**1564**	782.14	1589	17.60	1618	976.23	N/A	N/A	2561	23.47	2356	6.56
rail5	**174**	166.15	175	6.55	176	358.26	233	7.65	206	276.35	245	46.76
rail6	**182**	3.74	**182**	1.67	**182**	205.73	216	7.17	197	696.02	202	965.76
rail7	**211**	443.22	212	8.16	213	300.89	273	9.46	245	500.24	285	28.96

Table 4: Summary Results on DSP benchmarks

Benchmark	#inst.	*DomSAT*		*FastWMDS*		*SATLike*		*Loandra*		*Open-wbo*	
		#win	time	#win	time	#win	time	#win	time	#win	time
unweighted	65	**62**	441.75	16	324.87	1	19.88	4	143.35	0	N/A
weighted	65	**62**	583.10	2	499.92	1	920.99	2	115.13	0	N/A
No Reduction											
unweighted	65	**47**	318.51	17	281.62	2	111.27	7	116.58	0	N/A
weighted	65	**40**	588.58	13	819.56	8	734.58	5	125.17	0	N/A

For each solver on each instance family, we report the number of instances where the solver finds the best solution among all solvers in the table, denoted by "#win" and the mean time of doing so over such winning instances. 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.

Experiment Results on SCP instances

In this subsection, we compare DomSAT with state of the art solvers on SCP benchmarks. As we can see from Table 1 and 2, DomSAT and WCC achieved the same results on these benchmarks, while the other solvers perform much worse than DomSAT and WCC. Rail instances are large scale industrial instances (>10000 variables) and are considered difficult to solve. Experiment results on these instances are presented in Table 3. DomSAT- represent the version of DomSAT without the reduction procedure. DomSAT and DomSAT- outperform all other solvers on all these instances, and DomSAT performs better than DomSAT-, which indicates a dramatic improvement for solving large scale instances.

Experiment Results on DSP instances

We also compare DomSAT with state of the art solvers on DSP benchmarks. We present the summary results of comparing DomSAT and other solvers on all benchmarks (unweighted + weighted DSP benchmarks) in Table 4. It is encouraging to see that, for both weighted and unweighted DSP problems, DomSAT is much better than all other solvers. For 130 instances (unweighted + weighted), DomSAT find the best solution for 124 of them while this number is only 18, 2, 6, 0 for FastWMDS, SATLike, Loandra and Open-wbo respectively. These results show the strong performance of DomSAT for solving DSP. Furthermore, we can see that, without reduction procedure, our algorithm is still much better than other solvers.

Table 5: Summary Results on DSP benchmarks

Benchmark	*DomSAT*		*DomSAT_alt1*		*DomSAT_alt2*		*DomSAT_alt3*	
	#win	time	#win	time	#win	time	#win	time
STS	**4**	225.01	**4**	225.01	**4**	237.26	**4**	242.14
OR	**65**	0.29	**65**	0.29	64	0.42	**65**	0.47
Rail	**5**	423.20	0	N/A	4	514.56	4	326.25
we_DSP	**38**	687.72	10	546.78	21	718.27	9	543.80
unwe_DSP	**63**	441.01	4	38.32	31	47.30	23	105.60

Experimental Analysis on DomSAT

There are three main ideas in our algorithm DomSAT. Thus, in order to demonstrate the effectiveness of these three ideas in our algorithm, we conduct experiments to compare DomSAT with the three alternative versions.

- DomSAT_alt1: This alternative version does not utilize the reduction procedure to simplify instances.
- DomSAT_alt2: This alternative version does not utilize the CC strategy.
- DomSAT_alt3: This alternative version does not utilize the Aspiration Mode.

From Table 5, we can see that DomSAT outperforms three alternatives significantly on large scale instances, which shows that our heuristics play an important role in our algorithm. For STS and all OR instances, all alternatives perform as well as DomSAT does. The main reason is that these small instances are easy to solve for our algorithm structure.

Conclusion

In this work, we formulated SCP and DSP into weighted Partial Maximum Satisfiability and proposed several reduction rules to simplify these MaxSAT instances. We also designed a local search algorithm tailored for such instances. Experiments result showed that our algorithm is better than all state of the art algorithms for SCP, DSP and MaxSAT. The strong results show that exploiting the features of the instances from specific domains in MaxSAT solving would lead to more effective solvers. We would like to explore features of other combinatorial optimization problems to design MaxSAT solvers for them.

Acknowledgement

This work is supported by Youth Innovation Promotion Association, Chinese Academy of Sciences (No.2017150).

References

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

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

Aoun, B.; Boutaba, R.; Iraqi, Y.; and Kenward, G. W. 2006. Gateway placement optimization in wireless mesh networks with qos constraints. *IEEE Journal on Selected Areas in Communications* 24(11):2127–2136.

Bautista, J., and Pereira, J. 2006. Modeling the problem of locating collection areas for urban waste management. an application to the metropolitan area of barcelona. *Omega* 34(6):617–629.

Beasley, J. E., and Chu, P. 1996. A genetic algorithm for the set covering problem. *European Journal of Operational Research* 94(2):392–404.

Beasley, J. E. 1990. Or-library: distributing test problems by electronic mail. *Journal of the operational research society* 41(11):1069–1072.

Cai, S., and Su, K. 2011. Local search with configuration checking for SAT. In *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*, 59–66.

Cai, S.; Luo, C.; Thornton, J.; and Su, K. 2014. Tailoring local search for partial maxsat. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2623–2629.

Cai, S.; Luo, C.; Lin, J.; and Su, K. 2016. New local search methods for partial maxsat. *Artif. Intell.* 240:1–18.

Cai, S. 2015. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 747–753.

Caprara, A.; Fischetti, M.; Toth, P.; Vigo, D.; and Guida, P. L. 1997. Algorithms for railway crew management. *Math. Program.* 79:125–141.

Caprara, A.; Toth, P.; and Fischetti, M. 2000. Algorithms for the set covering problem. *Annals OR* 98(1-4):353–371.

Chalupa, D. 2018. An order-based algorithm for minimum dominating set with application in graph mining. *Inf. Sci.* 426:101–116.

Davies, J., and Bacchus, F. 2011. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, 225–239.

Demirovic, E., and Musliu, N. 2017. Maxsat-based large neighborhood search for high school timetabling. *Computers & OR* 78:172–180.

Demirovic, E.; Musliu, N.; and Winter, F. 2019. Modeling and solving staff scheduling with partial weighted maxsat. *Annals OR* 275(1):79–99.

Fulkerson, D.; Nemhauser, G. L.; and Trotter, L. 1974. Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of steiner triple systems. In *Approaches to integer programming*. 72–81.

Gao, C.; Weise, T.; and Li, J. 2014. A weighting-based local search heuristic algorithm for the set covering problem. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6-11, 2014*, 826–831.

Hedetniemi, S. M.; Hedetniemi, S. T.; Jacobs, D. P.; and Srimani, P. K. 2003. Self-stabilizing algorithms for minimal dominating sets and maximal independent sets. *Computers & Mathematics with Applications* 46(5-6):805–811.

Jacobs, L. W., and Brusco, M. J. 1995. Note: A local-search heuristic for large set-covering problems. *Naval Research Logistics (NRL)* 42(7):1129–1140.

Jiang, H.; Li, C.; Liu, Y.; and Manyà, F. 2018. A two-stage maxsat reasoning approach for the maximum weight clique problem. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 1338–1346.

Jovanovic; Raka; Milan; and Dana. 2010. Ant colony optimization applied to minimum weight dominating set problem. *Plant Physiology* 146(3):173–176.

Lei, Z., and Cai, S. 2018. Solving (weighted) partial maxsat by dynamic local search for SAT. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, 1346–1352.

Luo, C.; Cai, S.; Su, K.; and Huang, W. 2017. CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. *Artif. Intell.* 243:26–44.

Martins, R.; Manquinho, V. M.; and Lynce, I. 2014. Open-wbo: A modular maxsat solver,. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, 438–445.

Narodytska, N., and Bacchus, F. 2014. Maximum satisfiability using core-guided maxsat resolution. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2717–2723.

Nitash, C. G., and Singh, A. 2014. An artificial bee colony algorithm for minimum weight dominating set. In *2014 IEEE Symposium on Swarm Intelligence, SIS 2014, Orlando, FL, USA, December 9-12, 2014*, 313–319.

Shen, C., and Li, T. 2010. Multi-document summarization via the minimum dominating set. In *Proceedings of the 23rd international conference on computational linguistics*, 984–992.

Wang, Y.; Cai, S.; Chen, J.; and Yin, M. 2018. A fast local search algorithm for minimum weight dominating set problem on massive graphs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, 1514–1522.