Pure MaxSAT and Its Applications to Combinatorial Optimization via Linear Local Search

Shaowei Cai[1,2(✉)] and Xindi Zhang[1,2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
{caisw,zhangxd}@ios.ac.cn
[2] School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing, China

Abstract. Maximum Satisfiability (MaxSAT) is a general model for formulating combinatorial optimization problems. MaxSAT formulas encoded from different domains have different features, yet most MaxSAT solvers are designed for general formulas. This work considers an important subclass of MaxSAT, named as Pure MaxSAT, which characterizes a wide range of combinatorial optimization problems particularly subset problems. We design a novel local search method for Pure MaxSAT, which combines the idea of linear search and local search and is dubbed as linear local search. Our algorithm LinearLS significantly outperforms state of the art MaxSAT solvers on Pure MaxSAT instances, including instances from MaxSAT Evaluations and those encoded from three famous NP hard combinatorial optimization problems. Moreover, LinearLS outperforms state of the art algorithms for each tested combinatorial optimization problem on the popular benchmarks.

Keywords: Pure MaxSAT · Combinatorial optimization · Linear local search

1 Introduction

Maximum Satisfiability (MaxSAT) is an optimisation version of Boolean Satisfiability (SAT), and its general form contains both hard clauses and soft clauses, where the soft clauses can be unweighted or weighted. Solving such a MaxSAT instance involves finding a truth assignment that satisfies the hard clauses along with a maximum number (resp. weight) of soft clauses. MaxSAT is a natural model for formulating combinatorial optimization problems, and has been used to efficiently solve many combinatorial optimization problems that appear in industrial domains.

Most MaxSAT algorithms are developed for general purpose and focus on achieving better performance on a wide range of benchmarks which come from

H. Simonis (Ed.): CP 2020, LNCS 12333, pp. 90–106, 2020.
https://doi.org/10.1007/978-3-030-58475-7_6

diverse domains, and they are usually tested on benchmarks from MaxSAT Evaluations (MSEs). The most popular and effective approach for MaxSAT is the SAT-based approach [1,17,35], which reformulates the MaxSAT optimization problem into a sequence of SAT decision problems and solves them by iteratively calling a SAT solver. SAT-based MaxSAT algorithms can be divided into two types: *linear* [6,7,26,32] and *core-guided* [2,21,36,39]. SAT-based algorithms are essentially complete: they can prove the optimality of the solutions they find when they terminate. Some SAT-based solver such as the linear search ones and the hybrid ones [3,4], refine the upper bound during the search, and can be used for incomplete solving. SAT-based solvers have shown strong performance in the MSEs.

There has been growing interest in incomplete MaxSAT algorithms in recent years, with a surge of new methods at the two recent MSEs. A main incomplete approach for MaxSAT is local search, which aims to find high quality solutions quickly. Local search algorithms typically maintain a complete assignment and modify it iteratively by flipping the value of variables to quickly visit the search space and look for solutions of increasing quality. Local search for MaxSAT has witnessed significant progress during recent years [10,20,27,30]. Particularly, a dynamic local search algorithm named SATLike [27] is competitive with SAT-based solvers on solving unweighted industrial instances.

When solving combinatorial optimization problems by MaxSAT, most works utilize the general solvers off the shelf [5,13,14,22]. However, MaxSAT instances from different domains have their own characteristics, which we believe should be taken into account. Very limited works have been done on developing MaxSAT solvers for specific problems such as Maximum Weight Clique [16,23]. But such algorithms are limited to just one specific problem. An important fact is that many combinatorial optimization problems share the same feature when they are formulated in MaxSAT. Therefore, a significant direction is to develop effective algorithms for important subclasses of MaxSAT, which can have better performance than general MaxSAT algorithms while at the same time can be applied to a wide range of problems.

In this work, we introduce an important subclass of MaxSAT called Pure MaxSAT (PureMS for short), which characterizes a wide range of combinatorial optimization problems, particularly including subset problems. In fact, a considerable portion of the benchmarks in recent MSEs belong to this subclass. For solving PureMS, we propose a new search paradigm named linear local search, which is inspired by the great success of the linear SAT-based solvers. The core idea is that, whenever finding a better solution, the algorithm only visits assignments with strictly lower soft cost (i.e., with smaller weight of falsified soft clauses). Thus, every feasible solution visited during the search has a lower cost than previously found solutions. This is the first time that the idea of linear search is integrated to local search for MaxSAT, and our experiments show that the linear local search is powerful for solving PureMS formulas.

Our linear local search is a two-phase local search algorithm, which consists of two phases in each iteration. The first one is dedicated to decrease the soft cost,

while the second focuses on satisfying hard clauses, subject to keeping the soft cost lower than the cost of the previously found best solution. To improve the local search, we propose a variant of the Variable Neighbourhood Descent (VND) method [34]. VND is a variant of Variable Neighbourhood Search (VNS), which benefits from the advantages of large neighbourhoods without incurring a high time complexity of the search steps. VND employs small neighbourhoods until a local optimum is encountered, at which point the search process switches to a larger neighbourhood (corresponding to flipping more variables in one iteration in the context of MaxSAT), which might allow further improvement. Different from previous VND or VNS methods which only consider the number of elements to change values, we also take into account a structure parameter, i.e., the total degree of the chosen variables.

We carry out experiments to evaluate our algorithm dubbed LinearLS on a wide range of benchmarks, including all PureMS instances in recent MSEs, as well as the benchmarks from three famous combinatorial optimization problems, namely maximum clique (MaxClq), minimum vertex cover (MinVC) and set cover problem (SCP). Our results show that LinearLS is significantly better than state of the art MaxSAT solvers, including SAT-based and local search ones on all the benchmarks. Moreover, LinearLS outperforms state of the art algorithms for MaxClq, MinVC and SCP. Note that, our algorithm is general for combinatorial optimization problems that can be formulated as PureMS, while the competitors are tailored for each specific problem respectively.

The remainder of this paper is structured as follows. The next section introduces background knowledge. Section 3 introduces the Pure MaxSAT problem, and Sect. 4 presents the linear local search method. Experiment results are presented in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Preliminary

Given a set of n Boolean variables $X = \{x_1, x_2, ..., x_n\}$, a *literal* is either a variable x_i (positive literal) or its negation $\neg x_i$ (negative literal). The *polarity* of a positive literal is 1, while the polarity of a negative literal is 0. A *clause* is a disjunction of literals (i.e. $C_i = l_{i1} \vee l_{i2} \vee ... \vee l_{ij}$), and can be viewed as a set of literals. A *unit clause* is a clause with only one literal. A Conjunctive Normal Form (CNF) formula $F = C_1 \wedge C_2 \wedge ... \wedge C_m$ is a conjunction of clauses.

A mapping $\alpha : X \rightarrow \{0, 1\}$ is called an *assignment*, and a *complete assignment* is a mapping that assigns to each variable either 0 or 1. Given an assignment α, a clause is satisfied if it has at least one true literal, and is falsified if all its literals are false under α.

Given a CNF formula, the Maximum Satisfiability (MaxSAT) problem concerns about finding an assignment to satisfy the most clauses. In its general form, the clauses are divided into hard clauses and soft clauses, where the soft clauses can be unweighted or weighted, and the goal is to find an assignment that satisfies all hard clauses and maximizes the number (resp. weight) of satisfied soft clauses. Such formulas are referred to as (Weighted) Partial MaxSAT. Hereafter, when we say MaxSAT, we refer to this kind of formulas.

Given a MaxSAT formula F and an assignment α to its variables, two important sets are defined here.

- $H_f(\alpha) = \{c|c$ is a hard clause falsified under $\alpha\}$.
- $S_f(\alpha) = \{c|c$ is a soft clause falsified under $\alpha\}$.

The cost functions are defined below.

- the *hard cost* of α, denoted as $cost_h(\alpha)$, is the number of falsified hard clauses under α.
- the *soft cost* of α, denoted as $cost_s(\alpha)$, is the number (or total weight) of falsified soft clauses under α.
- the *cost* of α is $cost(\alpha) = +\infty \cdot cost_h(\alpha) + cost_s(\alpha)$.

An assignment α is feasible iff it satisfies all hard clauses in F. It is easy to see $cost(\alpha) = cost_s(\alpha)$ for feasible assignments, while $cost(\alpha) = +\infty$ for infeasible assignments.

Two variables are *neighbors* if they occur in at least one clause, and $N(x)$ denotes all the neighboring variables of x. The degree of x is denoted as $degree(x) = |N(x)|$. We use $\overline{\Delta}(F)$ to denote the averaged degree of formula F.

Below we give the definitions of the three combinatorial optimization problems studied in our experiments.

MaxClq and MinVC: Given an undirected graph $G = (V, E)$, a clique is a subset $K \subseteq V$ whose elements are pairwise adjacent, while a vertex cover is a subset $C \subseteq V$ such that every edge has at least one endpoint in C. Given a graph, the Maximum Clique (MaxClq) problem is to find a maximum sized clique, while the Minimum Vertex Cover (MinVC) problem is to find the minimum sized vertex cover.

SCP: Given an ground set U and a set S of subsets of U with $\cup_{\forall s \in S} = U$, where each element in S is associated with a weight $w(s)$, the goal of Set Cover Problem (SCP) is to find a set $F \subseteq S$ of the smallest total weight but still contains all elements in U, that is, $\cup_{\forall s \in F} = U$. In the unweighted version of SCP, also known as uniform cost SCP, each element in S has the same weight 1, and thus the goal is to find $F \subseteq S$ such that the cardinality of F is the smallest.

3 The Pure MaxSAT Problem

We propose a new variant of MaxSAT named Pure MaxSAT, which is an important subclass of the MaxSAT problem.

Definition 1 (pure clause). *A clause is a pure clause if all its literals are of the same polarity (either positive or negative). The polarity of a pure clause is defined as the polarity of its literals.*

Definition 2 (Pure MaxSAT). *The Pure MaxSAT problem is a special type of Partial MaxSAT where all hard clauses are pure clauses with the same polarity, and all soft clauses are pure clauses with the opposite polarity. In the weighted Pure MaxSAT, each soft clause has a positive number as its weight.*

When formulated as the language of MaxSAT, many combinatorial optimization problems naturally fall into the class of Pure MaxSAT. We give three examples, which are famous NP hard problems with wide applications of their own.

- MaxClq: For each vertex $i \in V$, the PureMS instance has a boolean variable x_i that indicates whether vertex i is chosen in the clique. For each vertex pair $(i, j) \notin E$ (E is the edge set), generate a hard clause $\neg x_i \vee \neg x_j$; for each vertex $i \in V$, generate a unit soft clause $\{x_i\}$.
- MinVC: For each vertex $i \in V$, the PureMS instance has a boolean variable x_i that indicates whether vertex i is chosen in the vertex cover. For each edge $(i, j) \in E$, generate a hard clause $x_i \vee x_j$; for each vertex $i \in V$, generate a unit soft clause $\{\neg x_i\}$.
- SCP: For each element (a subset) $s \in S$, the PureMS instance has a boolean variable x_s that indicates whether s is chosen in the solution. For each element $e \in U$, generate a hard clause $\{x_s | s \in S, e \in s\}$, to ensure that each element in U is covered by at least one subset in S. For each element $s \in S$, generate a soft clause $\{\neg x_s\}$ and its weight is equal to $w(s)$.

Observing the feature of PureMS, we can gain some insights on local search algorithms for this problem. Since the polarity of hard clauses is opposite to that of the soft clauses, the goal of satisfying more hard clauses and the goal of satisfying more soft clauses are obviously conflicting. Whenever we flip a variable to reduce $cost_s$, it causes an increase on $cost_h$, although sometimes the increment can be 0 (rarely happens). Similarly, a flip of a variable which reduces $cost_h$ potentially goes along with an increase on $cost_s$. This observation leads us to design linear local search for PureMS.

4 Linear Local Search for Pure MaxSAT

In this section, we propose a linear local search algorithm for PureMS. We firstly introduce the linear local search framework and the scoring function, and then present the algorithm.

4.1 Linear Local Search and Its Scoring Function

We propose a two-phase local search framework (Algorithm 1), which allows to implement a linear search that visits solutions with monotonically decreasing *cost*. First, we note that, for PureMS, since the polarity of literals in hard clauses are the same, we can produce a feasible initial assignment with guarantee. At the trivial case, we can simply assign 0 to all variables if hard clauses consist

of negative literals, and 1 on the contrary. Nevertheless, there are better initialization algorithms. After the initialization, the algorithm executes a loop until reaching a time limit. Each iteration of the loop consists of two phases.

Algorithm 1: A Linear Local Search Framework for PureMS

```
1  Input: MaxSAT instance F, the cutoff time
2  Output: A feasible assignment α*
3  begin
4      α ← InitAssignment();
5      while elapsed time < cutoff do
6          if H_f(α) = ∅ then
7              α* ← α, cost* ← cost(α);
8          flip some variables to decrease cost_s(α);
9          while H_f(α) ≠ ∅ do
10             choose a variable y from falsified hard clauses;
11             if flipping y would cause cost_s(α) ≥ cost* then break;
12             flip(α, y);
13     return (α*, cost*)
```

- In the first phase, the algorithm chooses some variables to flip, with the purpose of decreasing the soft cost. This phase produces some newly falsified hard clauses.
- In the second phase, the algorithm tries to satisfy as many hard clauses as possible, with a constraint that keeps the soft cost strictly lower than $cost^*$ (the cost of the best found solution). Thus, if all hard clauses are satisfied (i.e., $H_f(\alpha) = \emptyset$), that means a better solution is found.

Local search algorithms are typically guided by scoring functions, which are used to evaluate variables and critical in selecting the variable to flip. We design a scoring function which cooperates well with the linear local search framework.

Our scoring function is related to a clause weighting scheme. Most local search algorithms for MaxSAT employ constraint weighting techniques, which serve as a form of diversification. Our algorithm utilizes a clause weighting scheme that works on hard clauses (the details will be described in the LinearLS algorithm in Sect. 4.2). We associate each hard clause c with a positive integer weight $hw(c)$[1], which are initialized to 1 and updated during the search. Note that the weights of soft clauses are not changed in our algorithm. Our scoring function relies on two basic functions which are defined below.

[1] To distinguish with the weight of soft clauses $w(c)$ in the original formula, we use $hw(c)$ to denote the hard clause weight introduced by our method.

Definition 3 (hard score, soft score). *Given a MaxSAT formula and let α be a complete assignment, the hard score of a variable x w.r.t. α is defined as*

$$hscore(\alpha, x) = \sum_{c \in H_f(\alpha)} hw(c) - \sum_{c \in H_f(\alpha')} hw(c),$$

and the soft score of x w.r.t. α is defined as

$$sscore(\alpha, x) = cost_s(\alpha) - cost_s(\alpha'),$$

where α' differs from α only in the value of x.

In this work, the α in scoring functions always refers to the current assignment and can be omitted. Hence, $hscore(\alpha, x)$ and $sscore(\alpha, x)$ can be written as $hscore(x)$ and $sscore(x)$. Intuitively, $hscore(x)$ and $sscore(x)$ are the incremental changes in the objective for flipping x w.r.t. the current assignment.

Lemma 1. *Given any PureMS formula F, and α is a complete assignment to F, for any variable x, we have*

$$hscore(\alpha, x) \cdot sscore(\alpha, x) \leq 0.$$

Proof: According to the definition of PureMS, all clauses in F are pure clauses and the literals in hard clauses have the opposite polarity to those in soft clauses. Without loss of generality, let us assume the hard clauses contain only positive literals, and the soft clauses contain only negative literals. If $hscore(\alpha, x) > 0$, which indicates that the flip of x make at least one falsified hard clause become satisfied, then it must flip the value of x from 0 to 1. Such a $0 \rightarrow 1$ flip never makes any falsified soft clause become satisfied, as all soft clauses have negative literals. Therefore, $hscore(\alpha, x)$ and $sscore(\alpha, x)$ cannot be positive at the same time. □

Based on these two basic functions and Lemma 1, we derive a novel scoring function as follows.

Definition 4 (ratio score). *The ratio score of a variable x is defined as*

$$rscore(x) = \frac{hscore(x)}{|sscore(x)| + 1}.$$

This *rscore* measures the ratio of *hscore* and *sscore*. We add one to the denominator to avoid the "divide by 0" error. Also, we adopt the absolute value of *sscore* for convenient usage—by doing this, we can prefer larger *rscore* no matter in the first or second phase. In the first phase, we focus on satisfying soft clauses, and the chosen variables have $sscore(x) > 0$ and $hscore(x) \leq 0$, and thus $rscore(x) \leq 0$. For equal *hscore*, a larger *rscore* means a larger *sscore*, which means satisfying more soft clauses; for equal *sscore*, a larger *rscore* means breaking fewer hard clauses. In the second phase, we focus on satisfying hard clauses, and the chosen variables have $hscore(x) > 0$ and $sscore(x) \leq 0$, and thus

$rscore(x) \geq 0$. For equal $sscore$, a larger $rscore$ means a larger $hscore$, which leads to more satisfied hard clauses; for equal $hscore$, a larger $rscore$ means a smaller $|sscore|$, which means breaking fewer soft clauses. Our algorithm employs $rscore$ in both phases of local search, and prefers to pick the variables with larger $rscore$.

4.2 The LinearLS Algorithm

Based on the linear local search framework and the $rscore$ function, we develop an algorithm named LinearLS (Algorithm 2). The algorithm is described in details below.

Initialization: Unlike previous local search algorithms for SAT/MaxSAT which generate the initial solution randomly, our algorithm employs a greedy strategy. Firstly, all variables are assigned with the value equal to the polarity of the soft clauses. This makes all hard clauses falsified and all soft clauses satisfied. Then, the algorithm iteratively picks a random falsified hard clause and flips a variable with highest $hscore$ in the clause, until there is no falsified hard clause. Thus the initial assignment is feasible. At the worst case, all variables are flipped in order to make all hard clauses satisfied, then the cost would be the largest among feasible solutions as all soft clauses are falsified in this situation. In practice, however, this initialization procedure usually finds a much better solution than the worst case.

After initialization, the local search loop (lines 5–20) is executed until a given cutoff time is reached. During the search, the best found solution α^* and its cost are updated. An important feature of our linear local search algorithm is that, whenever we find a feasible solution, we are sure that it is better than α^*, as the algorithm always keeps $cost_s(\alpha)$ strictly lower than $cost^*$. Thus, whenever α becomes feasible, α^* is updated to α and $cost^*$ is updated accordingly (lines 6–7). When the algorithm reaches the time limit, it returns the best found solution α^* and its cost.

The local search is based on the two-phase framework, and we propose a variant of Variable Neighbourhood Descent (VND) method for striking a good balance between exploitation and exploration.

In the first phase (lines 8–13), we flip K variables, where K is adjusted according to the algorithm's behavior. If the algorithm has not found a better solution for a long period (which is set to $2{\cdot}10^4$ steps for SCP benchmarks and 10^4 for the others in LinearLS), then K increases by 1 for improving exploration, while once the algorithm finds a solution, K is reset to 1 for fast converge. This is implemented in the $Adjust\_flip\_num\_phase1(K)$ function. Each flipping variable in the first phase is chosen from all falsified soft clauses by picking the variable with the highest $rscore$ (line 10), breaking ties by preferring the one that is least recently flipped. Additionally, we set a dynamic maximum limit to K by considering the total degree of the flipping variables in the first phase (line 12). Once this value achieves a threshold ($t \times \overline{\Delta}(F)$, where t is set to 1 for

Algorithm 2: LinearLS

```
1  Input: Pure MaxSAT instance F, the cutoff time
2  Output: A feasible assignment α* and its cost
3  begin
4      α ← InitSolution();
5      while elapsed time < cutoff do
6          if H_f(α) = ∅ then
7              α* ← α, cost* ← cost(α);
8          SumDegree ← 0;
9          for i ← 1 to K do
10             x ← a variable in S_f(α) with highest rscore;
11             flip(α, x);
12             SumDegree+ = degree(x);
13             if SumDegree ≥ t × Δ̄(F) then break ;
14         while H_f(α) ≠ ∅ do
15             c ← a random falsified hard clause;
16             y ← a variable in c with highest rscore;
17             if cost_s(α) − sscore(α, y) ≥ cost* then break ;
18             flip(α, y);
19         Adjust_flip_num_phase1(K);
20         Update_hard_clause_weights();
21     return (α*, cost*)
```

MaxClq and SCP benchmarks, 2 for the rest), the first phase is stopped and the algorithm goes to the second phase (line 13).

Here we provide the intuition of limiting VND with a degree based upper bound. Generally, the more variables flipped in the first phase, the more candidate variables are generated for the second phase. However, the other factor to the number of candidate flipping variables (thus the size of search area) in the second phase is the degree of the variables flipped in the first phase. We take into account both factors in our VND method.

The second phase (lines 14–18) is dedicated to satisfy hard clauses, and thus each flipping variable is chosen from a random falsified hard clause. The variable with highest $rscore$ is picked, breaking ties by preferring the one that is least recently flipped. For each selected variable, LinearLS checks whether its flip would cause $cost_s(\alpha)$ greater than or equal to $cost^*$, and if this is the case, it leaves the second phase immediately without flipping the variable. By doing this, we guarantee that $cost_s(\alpha) < cost^*$ always holds during the search.

In the end of each iteration, the K value is updated when necessary according to our VND method. Also, the hard clause weights are updated (line 20): the weight of each falsified hard clause is increased by 1, and when the average weight achieves a threshold (which is set to $\frac{n}{2}$), early weighting decisions are forgotten as $hw(c) \leftarrow \rho \cdot hw(c)$, where $\rho \in (0, 1)$ is a constant factor and set to 0.3.

4.3 More Optimizations

An effective strategy to avoid the cycle phenomenon (i.e., revisiting some search areas) in local search is the Configuration Checking (CC) strategy [12], which forbids flipping a variable x, if after the last time x was flipped, none of its neighboring variables has changed its value. The CC strategy has proved effective in local search for SAT [11] and MaxSAT [30,31]. LinearLS also employs CC. Variables flipped in last iteration are also forbidden to be flipped again. These are common techniques in local search to reduce the cycle phenomenon.

5 Experiments

We carry out experiments to compare LinearLS with state of the art algorithms on a wide range of benchmarks. LinearLS is implemented in C++ and compiled by g++ with -O3 option. Our all experiments were conducted on a server using Intel Xeon Platinum 8153 @2.00 GHz, 512G RAM, running Centos 7.7.1908 Linux operation system. The time limit for all algorithms is 300 s, except that we additionally test an exact MaxSAT solver for one hour.

5.1 Results on PureMS Benchmarks from MSEs

We collect all PureMS instances from both unweighted and weighted benchmarks in MaxSAT Evaluations (MSEs) 2017, 2018 and 2019. There are several duplicate instances in the unweighted benchmarks of the three MSEs. We compare LinearLS with 4 state of the art MaxSAT solvers, from which are 1 local search solver and 3 SAT-based solvers.

Table 1. Results on Pure MaxSAT benchmarks from MaxSAT Evaluations 2017–2019, including unweighted benchmarks and weighted benchmarks.

Benchmark	#inst.	*LinearLS*		*SATLike(_w)*		*Loandra*		*TT-OpenWBO*		*RC2*		*RC2(1h)*	
		#win	Time	#win	Time	#win	Time	#win	Time	#win	Time	#win	Time
Unweighted													
MSE17	113	**111**	6.1	48	31.6	57	27.5	57	40.9	57	15.1	64	153.9
MSE18	110	**100**	16.2	46	29.5	46	23.7	36	44.1	61	56.0	70	235.2
MSE19	101	**88**	22.8	28	17.0	33	51.6	25	59.4	42	84.2	49	264.4
MSEall	284	**261**	14.2	112	29.6	121	28.0	105	41.9	148	50.4	168	206.7
Weighted													
MSE17	40	**40**	<0.1	31	0.1	33	16.7	20	0.1	25	25.5	37	372.8
MSE18	15	**15**	<0.1	15	0.1	14	29.5	0	N/A	4	151.7	14	859.2
MSE19	51	**51**	30.7	30	6.1	21	10.7	8	27.4	19	45.7	25	238.7
MSEall	106	**106**	14.8	76	2.5	68	17.5	28	7.9	48	44.0	76	418.3

- SATLike [27] is the best local search MaxSAT solver, which won the two unweighted categories of incomplete track in MSE 2018 and placed 2nd in the 300 s unweighted category of incomplete track in MSE 2019. SATLike has another version optimized for weighted categories, which is denoted as SATLike_w.
- Loandra [6] won the two unweighted categories, and was ranked 2nd in two weighted categories of incomplete track in MSE 2019.
- TT-Open-WBO-inc [38] won the two weighted categories of incomplete track in MSE 2019.
- RC2 (implementing the relaxable cardinality constraints method) [37] won both weighted and unweighted categories of complete track in MSE 2019. Since RC2 is an exact solver, in our experiments, we test RC2 with 2 time limits, 300 s (as with other solvers) and one hour. We note that local search and exact solvers have different advantages and it is better to see them as complementary alternatives. The comparisons with exact solvers are just for reference, which may give us some insights.

Table 2. Averaged SCORE results of MaxSAT solvers on each family of MSE benchmarks. We also report the results of the complete solver RC2 with 1 h time limit for reference.

Domain(#inst.)	*LinearLS*	*Linear_init*	*SATLike(_w)*	*Loandra*	*TTOpenWBO*	*RC2*	*RC2*(1h)
Unweighted							
maxclique(68)	**1.0**	0.978	0.995	0.995	0.997	0.441	0.485
aes(14)	**1.0**	0.882	0.895	0.769	0.303	0.143	0.143
frb(40)	**1.0**	0.978	0.994	0.999	**1.0**	0.975	**1.0**
bcp-syn(53)	**1.0**	0.469	0.996	0.955	0.92	0.396	0.509
optic(69)	**1.0**	0.904	0.989	0.961	0.835	0.333	0.377
drmx-cryptogen(40)	0.991	0.827	0.984	0.955	0.881	0.825	**1.0**
Weighted							
auc-paths(35)	**1.0**	0.98	**1.0**	**1.0**	0.993	0.257	0.886
auc-scheduling(20)	**1.0**	0.996	0.996	0.999	**1.0**	**1.0**	**1.0**
MinimumWeight DominatingSetProblem(7)	**1.0**	0.966	0.402	0.708	0.705	0.0	0.0
auctions(16)	**1.0**	0.985	0.996	0.996	0.995	0.312	0.625
set-covering(28)	**1.0**	0.888	0.996	0.981	0.949	0.5	0.536

For each algorithm on each instance family, we report the number of instances where the solver finds the best solution among all solvers ("#win") and the mean time of doing so over such winning instances. These results (Table 1) clearly show the superiority of LinearLS over other MaxSAT solvers. Particularly, the "#win" number of LinearLS is always significantly larger than other solvers.

To show how far the solution provided by a solver are from the best solution found by all the solvers, for each algorithm $\mathcal{A}$ on each fomula F, we calculate a metric measured as $SCORE(F, \mathcal{A}) = \frac{BEST\_COST(F)+1}{COST(F,\alpha^{\mathcal{A}})+1}$, where $\alpha^{\mathcal{A}}$ is the solution found by algorithm $\mathcal{A}$ while $BEST\_COST(F)$ denotes the lowest cost found in

the time limit by any of the solvers. These benchmarks consists formulas encoded from different domains, and we report the averaged *SCORE* for each algorithm on each domain in Table 2. The *SCORE* of LinearLS is 1.0 (full score) for all domains except 0.991 for the *cryptogen* domain. Nevertheless, the best SCORE is obtained by RC2 for one hour time limit. If we compare all the solvers with the time limit of 300 s, then LinearLS is still the best, achieving a full score, which indicates its strong performance on a wide range of benchmarks from diverse domains.

We also calculate the *SCORE* of the initial solutions of LinearLS. As can be seen from the table, the initial solutions of LinearLS are better than the solutions returned by TTOpenWBO and RC2 on most of the Pure MaxSAT instances. Besides, although the initial solutions are not as good as those returned by the incomplete solvers SATLike and Loandra, the gaps are not large. This indicates that the design of the problem is an important factor to the good performance on Pure MaxSAT. By executing the local search procedure of LinearLS, the solutions are further improved.

Table 3. Results on MaxClq benchmarks. This table reports results for three MaxClq benchmarks, including Kidney Exchange (Kidney), Research Excellence Network (REN) and DIMACS. The error-correcting codes (ECC) benchmark instances are too easy that all algorithms find the optimal solution quickly and not reported.

Solvers	Kidney(120)		REN(129)		DIMACS(37)		MaxClq_all(286)	
	Win	Time	Win	Time	Win	Time	Win	Time
LinearLS	**120**	0.1	**129**	**0.1**	**33**	11.1	**282**	1.4
LSCC	118	0.8	127	<0.1	**33**	**<0.1**	278	0.3
BBMS	98	27.4	**129**	4.8	26	9.3	253	14.0
IncMaxCLQ	106	36.9	110	5.2	26	32.1	242	22.0
IncMC2	110	3.4	107	<0.1	26	10.9	243	2.7
MaxClqDyn	74	3.7	114	0.6	21	15.6	209	3.2
MCS	108	2.8	115	0.6	20	13.6	243	2.7
SATLike	84	13.3	115	10.6	8	19.0	207	12.0
Loandra	88	5.3	101	2.4	16	19.7	205	5.0
TT-OpenWBO	86	16.5	102	8.3	17	56.6	205	15.8

5.2 Results on Maximum Clique Benchmarks

We evaluate LinearLS on 4 popular MaxClq benchmarks which are mostly from applications [33]:

– Kidney Exchange, where the clique stands for a maximally desirable set of donor/patient exchanges. The instances were generated using data from [15].

- Error-correcting Codes (ECC), where the clique stands for a set of words maximally pair-wise distant [40].
- Research Excellence Network (REN) [33], where the clique stands for the optimal set of publications that a university department can provide to the authority assessing it.
- DIMACS, the MaxClq benchmark from Second DIMACS Implementation Challenge (1992–1993)[2]. Thirty seven graphs were selected by the organizers to be the Second DIMACS Challenge Test Problems.

Besides the MaxSAT algorithms, we compare with the following MaxClq algorithms. According to [28], state of the art MaxClq algorithms include IncMC2 [28], BBMC [42,43], IncMaxCLQ [29], MCS[44], MaxCliqueDyn [25]. We also compare with LSCC [46], which is a recent local search algorithm that performs well on both unweighted and weighted MaxClq benchmarks.

The ECC instances are so easy that all algorithms find the optimal solution quickly, and the local search algorithms do so within one second, and thus are not reported. The results (Table 3) show that Our LinearLS gives the best performance in terms of the solution quality, and is the best algorithm for the two application benchmarks namely Kidney and REN. Although the other local search LSCC is fast, it fails to find the best solution for some instances in these two benchmarks. The MaxSAT solvers, including SATLike, Loandra and TT-OpenWBO, perform much worse than LinearLS.

5.3 Results on Minimum Vertex Cover Benchmark

Recently, MinVC algorithms focus on solving massive graphs. Particularly, the Network Repository [41], which collects massive graphs from various areas, has become the most popular benchmark for testing MinVC algorithms in recent years [8,9,24,45]. FastVC [8] is an efficient local search for massive graphs, and afterwards it is improved by a preprocessor, resulting in the FastVC2+p algorithm [9]. Seen from the literature, FastVC2+p is currently the best algorithm for solving MinVC on the Network Repository instances.

On these massive MinVC instances, all the MaxSAT solvers perform significantly worse than LinearLS on almost all instances, and thus we do not report their results here. We focus on the comparison with MinVC algorithms FastVC and FastVC2+p. For fair comparison, when compared with FastVC2+p, LinearLS also utilizes the preprocessor in FastVC2+p to preprocess the graphs. We choose the graphs with at least 10^5 vertices, resulting in 65 graphs. Each algorithm is performed 10 runs on each graph with random seed from 1 to 10. We report the number of instances where the algorithm gives better results in terms of the minimum cost ('min') and the averaged cost ('avg') among the 10 runs. Seen from Table 4, the performance of LinearLS is surprisingly good on these massive MinVC instances, pushing the state of the art in MinVC solving on massive graphs.

[2] ftp://dimacs.rutgers.edu/pub/challenges.

Table 4. Results on large MinVC instances in Network Repository

	LinearLS		*FastVC*	
	Min	Avg	Min	Avg
#better-solution	**42**	**42**	12	13
#equal-solution	11	10	11	10
	LinearLS+p		*FastVC2+p*	
	Min	Avg	Min	Avg
#better-solution	**22**	**27**	17	14
#equal-solution	26	24	26	24

5.4 Results on Set Cover Benchmarks

We evaluate LinearLS on 2 important SCP benchmarks, including (1) the STS benchmark [18], which contains unweighted SCP instances from Steiner triple systems; (2) the Rail benchmark[3], which contains weighted SCP instances that arise from an application in Italian railways and have been contributed by Paolo Nobili.

LinearLS is compared with SATLike(_w) and the best SAT-based solver for each SCP benchmark. Also, it is compared with the SCP algorithm from [19], which is the best SCP algorithm on both unweighted and weighted SCP (the algorithm was not given a name, and is denoted by the paper [19]). Since the number of instances is limited, each algorithm is performed 10 runs on each instance, and we report the minimum cost and averaged cost, and the aver-

Table 5. Results on unweighted and weighted SCP instances

Instance	*LinearLS*		[19]		*SATLike*		*Loandra*	
	Min(avg)	Time	Min(avg)	Time	Min(avg)	Time	Min(avg)	Time
STS135	**103(103.0)**	3.3	**103(103.0)**	3.7	104(104.0)	292.6	104(104.0)	62.3
STS243	**198(198.0)**	<0.1	**198(198.0)**	0.1	198(201.8)	285.4	202(202.0)	258
STS405	**335(335.0)**	3.8	**335**(335.8)	31.0	342(344.0)	288.4	347(347.0)	4.9
STS729	**617(617.0)**	7.0	**617**(619.0)	26.5	646(647.4)	270.3	643(643.0)	5.7
	LinearLS		[19]		*SATLike_w*		*TT-OpenWBO*	
rail507	**176(176.0)**	3.6	**176**(176.3)	101.5	194(196.0)	114.4	248(248.0)	64.8
rail516	**182(182.0)**	33.8	**182**(182.1)	128.8	188(189.4)	170.4	226(226.0)	21.1
rail582	**213(213.0)**	1.9	**213**(213.9)	136.4	223(225.2)	176.5	286(286.0)	25.4
rail2536	**716(716.0)**	93.3	764(772.1)	288.1	N/A(N/A)	N/A	1125(1125.0)	3.9
rail2586	**990(990.0)**	30.2	1036(1055.0)	292.8	N/A(N/A)	N/A	1463(1463.0)	3.2
rail4284	**1117(1117.0)**	173.8	1203(1221.8)	287.5	N/A(N/A)	N/A	1734(1734.0)	4.1
rail4872	**1589(1589.0)**	55.8	1688(1733.2)	295.0	N/A(N/A)	N/A	2355(2355.0)	3.5

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

aged run time to find the final solution in each run. The results (Table 5) show that LinearLS outperforms the MaxSAT competitors significantly. Moreover, LinearLS finds better solutions than the SCP algorithm [19] and is much faster.

6 Conclusions

We introduced the Pure MaxSAT problem, which is an important subclass of MaxSAT and characterizes many combinatorial optimization problems particularly subset problems. We proposed the linear local search method for PureMS, which is the first work exploiting linear search in local search for MaxSAT problems. Experiments on benchmarks from MaxSAT Evaluations and benchmarks of three famous NP hard problems showed that our algorithm significantly outperforms previous MaxSAT algorithms, and achieves better results than state of the art specific algorithms for the three problems.

It is interesting to develop exact algorithms for Pure MaxSAT that can achieve better results than general MaxSAT solvers. Also, we would like to study the inference rules and reduction rules for Pure MaxSAT, which can be used to further improve the performance of Pure MaxSAT solvers.

Acknowledgments. This work is partially supported by Youth Innovation Promotion Association of Chinese Academy of Sciences [No. 2017150] and Beijing Academy of Artificial Intelligence (BAAI).

References

1. Ansótegui, C., Bonet, M.L., Levy, J.: SAT-based MaxSAT algorithms. Artif. Intell. **196**, 77–105 (2013)
2. Ansótegui, C., Didier, F., Gabàs, J.: Exploiting the structure of unsatisfiable cores in MaxSAT. In: Proceedings of IJCAI 2015, pp. 283–289 (2015)
3. Ansótegui, C., Gabàs, J.: WPM3: an (in)complete algorithm for weighted partial MaxSAT. Artif. Intell. **250**, 37–57 (2017)
4. Ansótegui, C., Gabàs, J., Levy, J.: Exploiting subproblem optimization in SAT-based MaxSAT algorithms. J. Heuristics **22**(1), 1–53 (2016). https://doi.org/10.1007/s10732-015-9300-7
5. Benedetti, M., Mori, M.: On the use of Max-SAT and PDDL in RBAC maintenance. Cybersecurity **2**(1) (2019). Article number: 19. https://doi.org/10.1186/s42400-019-0036-9
6. Berg, J., Demirović, E., Stuckey, P.J.: Core-boosted linear search for incomplete MaxSAT. In: Rousseau, L.-M., Stergiou, K. (eds.) CPAIOR 2019. LNCS, vol. 11494, pp. 39–56. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19212-9_3
7. Berre, D.L., Parrain, A.: The SAT4J library, release 2.2. JSAT **7**(2–3), 59–64 (2010)
8. Cai, S.: Balance between complexity and quality: local search for minimum vertex cover in massive graphs. In: Proceedings of IJCAI 2015, pp. 747–753 (2015)
9. Cai, S., Lin, J., Luo, C.: Finding a small vertex cover in massive sparse graphs: construct, local search, and preprocess. J. Artif. Intell. Res. **59**, 463–494 (2017)

10. Cai, S., Luo, C., Thornton, J., Su, K.: Tailoring local search for partial MaxSAT. In: Proceedings of AAAI 2014, pp. 2623–2629 (2014)
11. Cai, S., Su, K.: Local search for Boolean Satisfiability with configuration checking and subscore. Artif. Intell. **204**, 75–98 (2013)
12. Cai, S., Su, K., Sattar, A.: Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif. Intell. **175**(9–10), 1672–1696 (2011)
13. Demirovic, E., Musliu, N.: MaxSAT-based large neighborhood search for high school timetabling. Comput. OR **78**, 172–180 (2017)
14. Demirovic, E., Musliu, N., Winter, F.: Modeling and solving staff scheduling with partial weighted MaxSAT. Ann. OR **275**(1), 79–99 (2019). https://doi.org/10.1007/s10479-017-2693-y
15. Dickerson, J.P., Procaccia, A.D., Sandholm, T.: Optimizing kidney exchange with transplant chains: theory and reality. In: AAMAS 2012, pp. 711–718 (2012)
16. Fang, Z., Li, C., Xu, K.: An exact algorithm based on MaxSAT reasoning for the maximum weight clique problem. J. Artif. Intell. Res. **55**, 799–833 (2016)
17. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006). https://doi.org/10.1007/11814948_25
18. Fulkerson, D.R., Nemhauser, G.L., Trotter, L.: Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems. In: Balinski, M.L. (ed.) Approaches to Integer Programming. MATHPROGRAMM, vol. 2, pp. 72–81. Springer, Heidelberg (1974). https://doi.org/10.1007/BFb0120689
19. Gao, C., Weise, T., Li, J.: A weighting-based local search heuristic algorithm for the set covering problem. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, pp. 826–831 (2014)
20. Guerreiro, A.P., Terra-Neves, M., Lynce, I., Figueira, J.R., Manquinho, V.: Constraint-based techniques in stochastic local search MaxSAT solving. In: Schiex, T., de Givry, S. (eds.) CP 2019. LNCS, vol. 11802, pp. 232–250. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30048-7_14
21. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proceedings of AAAI 2011 (2011)
22. Huang, W., et al.: Finding and proving the exact ground state of a generalized Ising model by convex optimization and MAX-SAT. Phys. Rev. B **94**, 134424 (2016)
23. Jiang, H., Li, C., Liu, Y., Manyà, F.: A two-stage MaxSAT reasoning approach for the maximum weight clique problem. In: Proceedings of AAAI 2018, pp. 1338–1346 (2018)
24. Katzmann, M., Komusiewicz, C.: Systematic exploration of larger local search neighborhoods for the minimum vertex cover problem. In: Proceedings of AAAI 2017, pp. 846–852 (2017)
25. Konc, J., Janezic, D.: An improved branch and bound algorithm for the maximum clique problem. Commun. Math. Comput. Chem. **58**, 569–590 (2007)
26. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: a partial Max-SAT solver. JSAT **8**(1/2), 95–100 (2012)
27. Lei, Z., Cai, S.: Solving (weighted) partial MaxSAT by dynamic local search for SAT. In: Proceedings of IJCAI 2018, pp. 1346–1352 (2018)
28. Li, C., Fang, Z., Jiang, H., Xu, K.: Incremental upper bound for the maximum clique problem. INFORMS J. Comput. **30**(1), 137–153 (2018)
29. Li, C., Fang, Z., Xu, K.: Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In: ICTAI 2013, pp. 939–946 (2013)

30. Luo, C., Cai, S., Su, K., Huang, W.: CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. Artif. Intell. **243**, 26–44 (2017)
31. Luo, C., Cai, S., Wu, W., Jie, Z., Su, K.: CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans. Comput. **64**(7), 1830–1843 (2015)
32. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: a modular MaxSAT solver'. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 438–445. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_33
33. McCreesh, C., Prosser, P., Simpson, K., Trimble, J.: On maximum weight clique algorithms, and how they are evaluated. In: Beck, J.C. (ed.) CP 2017. LNCS, vol. 10416, pp. 206–225. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66158-2_14
34. Mladenovic, N., Hansen, P.: Variable neighborhood search. Comput. OR **24**(11), 1097–1100 (1997)
35. Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and core-guided MaxSAT solving: a survey and assessment. Constraints Int. J. **18**(4), 478–534 (2013). https://doi.org/10.1007/s10601-013-9146-2
36. Morgado, A., Heras, F., Marques-Silva, J.: Improvements to core-guided binary search for MaxSAT. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 284–297. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_22
37. Morgado, A., Ignatiev, A., Marques-Silva, J.: MSCG: robust core-guided MaxSAT solving. JSAT **9**, 129–134 (2014)
38. Nadel, A.: Tt-Open-WBO-Inc.: Tuning polarity and variable selection for anytime SAT-based optimizatio. In: Proceedings of MaxSAT Evaluation 2019: Solver and Benchmark Description, p. 29 (2019)
39. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: Proceedings of AAAI 2014, pp. 2717–2723 (2014)
40. Östergård, P.R.J.: A new algorithm for the maximum-weight clique problem. Electron. Notes Discrete Math. **3**, 153–156 (1999)
41. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: Proceedings of AAAI 2015, pp. 4292–4293 (2015)
42. Segundo, P.S., Lopez, A., Batsyn, M., Nikolaev, A., Pardalos, P.M.: Improved initial vertex ordering for exact maximum clique search. Appl. Intell. **45**(3), 868–880 (2016). https://doi.org/10.1007/s10489-016-0796-9
43. Segundo, P.S., Rodríguez-Losada, D., Jiménez, A.: An exact bit-parallel algorithm for the maximum clique problem. Comput. OR **38**(2), 571–581 (2011)
44. Tomita, E., Sutani, Y., Higashi, T., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments. IEICE Trans. **96-D**(6), 1286–1298 (2013)
45. Wagner, M., Friedrich, T., Lindauer, M.: Improving local search in a minimum vertex cover solver for classes of networks. In: IEEE Congress on Evolutionary Computation, CEC 2017, pp. 1704–1711 (2017)
46. Wang, Y., Cai, S., Yin, M.: Two efficient local search algorithms for maximum weight clique problem. In: Proceedings of AAAI 2016, pp. 805–811 (2016)