



LS-DTKMS: A Local Search Algorithm for Diversified Top- k MaxSAT Problem

Junping Zhou  

School of Information Science and Technology, Northeast Normal University, Changchun, China

Jiixin Liang  

School of Information Science and Technology, Northeast Normal University, Changchun, China

Minghao Yin¹  

School of Information Science and Technology, Northeast Normal University, Changchun, China
Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun, China

Bo He 

School of Information Science and Technology, Northeast Normal University, Changchun, China

Abstract

The Maximum Satisfiability (MaxSAT), an important optimization problem, has a range of applications, including network routing, planning and scheduling, and combinatorial auctions. Among these applications, one usually benefits from having not just one single solution, but k diverse solutions. Motivated by this, we study an extension of MaxSAT, named Diversified Top- k MaxSAT (DTKMS) problem, which is to find k feasible assignments of a given formula such that each assignment satisfies all hard clauses and all of them together satisfy the maximum number of soft clauses. This paper presents a local search algorithm, LS-DTKMS, for DTKMS problem, which exploits novel scoring functions to select variables and assignments. Experiments demonstrate that LS-DTKMS outperforms the top- k MaxSAT based DTKMS solvers and state-of-the-art solvers for diversified top- k clique problem.

2012 ACM Subject Classification Theory of computation → Random search heuristics; Theory of computation → Constraint and logic programming

Keywords and phrases Top- k , MaxSAT, local search

Digital Object Identifier 10.4230/LIPIcs.SAT.2023.29

Supplementary Material *Dataset:* <https://github.com/LyreRabbit/DTKMS>

archived at `swh:1:dir:d898c27229263839aaeef82ca95094c2f6643f11`

Funding This work is supported by Science and Technology Development Program of Jilin Province (20230101060JC and YDZJ202201ZYTS412), NSFC (under Grant No.61976050, 61972384), the Fundamental Research Funds for the Central Universities 2412019ZD013, and Jilin Science and Technology Department 20200201280JC.

Acknowledgements We would like to thank all the anonymous reviewers for their helpful comments.

1 Introduction

The Maximum Satisfiability (MaxSAT), an optimization version of the famous Satisfiability (SAT) problem, concerns about finding an assignment to satisfy all hard clauses as well as the maximum number of soft clauses by given a general form of propositional formula, which is represented as the Conjunctive Normal Form (CNF) containing both hard and soft clauses. Recently, the success of the MaxSAT algorithms [9, 11, 12, 17, 35] has contributed

¹ Corresponding Author



to a significant number of applications, such as network routing [20], planning and scheduling [27], combinatorial auctions [4], software engineering [13], data analysis [8] and machine learning [19]. Among these real-life applications, one actually benefits from not just one solution but diverse solutions. For example, in a social network, mining diverse communities often helps to find different topics and reduce the amount of overlapping information [7]. In robotic motion planning, practitioners are often interested in finding diverse paths because some pre-computed paths sometimes become invalid due to the relocation of obstacles or the reconfiguration of the robot [31]. In natural language understanding, machine translation systems benefit from working with multiple plausible parses of a sentence because sentences are often ambiguous [15]. In computational biology, computing multiple configurations of a protein structure is believed to be helpful to assess the sensitivity of the model [38]. Thus, in order to better meet the users' needs in numerous applications, it is essential to determine diverse solutions for MaxSAT problem.

This problem is introduced as the Diversified Top- k MaxSAT (DTKMS) problem, which is equivalent to MaxSAT when $k = 1$. Given a CNF formula, the objective of DTKMS is to determine at most k feasible assignments such that each assignment satisfies all hard clauses and the k assignments together satisfy the maximum number of soft clauses. As an extension of the MaxSAT problem, DTKMS is a bi-standard optimization problem that balances correlation and diversity of results. Now, a straightforward method for solving DTKMS involves first exhaustively searching for all feasible assignments and then employing the max k -coverage algorithm. Unfortunately, this method is not practical because the enumeration consumes enormous amounts of time and memory. Thus, to effectively solve DTKMS, two major challenges should be tackled. (1) As the number of feasible assignments for MaxSAT is potentially exponential, how can the diversification requirement be met without generating highly overlapping assignments? (2) Without an exhaustive search, how can the quality of the solutions be guaranteed?

In this work, for replying the above challenges, we propose a local search algorithm for the DTKMS problem, named LS-DTKMS, which features new scoring functions, including *score of variable* and *score of assignment*. To avoid generating highly overlapping assignments, we design a new *score of variable* scheme, unlike the previous one that only works on the current assignment. We evaluate the effect of flipping variables on both the current assignment and the DTKMS solution containing k assignments. By applying the new scheme in our variable selection heuristic, LS-DTKMS is able to generate diversified assignments. In addition, we design a *score of assignment* scheme to estimate the quality of a feasible assignment. After combining it with a key solution updating rule, LS-DTKMS can achieve a guaranteed approximation ratio of 0.25 if the final solution is obtained according to the updating rule. We conduct experiments to compare LS-DTKMS against top- k MaxSAT based DTKMS solvers on top- k MaxSAT instances from the MaxSAT Evaluation (MSE) 2020 top- k track. The results demonstrate that LS-DTKMS significantly outperforms the other solvers. To further verify the effectiveness of our algorithm, we apply LS-DTKMS to Diversified Top- k Clique Search (DTKCS) problem, which is to find k maximal cliques such that they cover the maximum number of vertices in a given graph [37]. The experimental results show that LS-DTKMS has better improvement than the state-of-the-art DTKCS solvers.

The outline of the paper is as follows. We first present some related definitions and a framework of top- k MaxSAT based DTKMS algorithm. In Section 3, we propose a local search algorithm and discuss the details of the techniques involved. In Section 5, we show the experimental results. Finally, we conclude the paper.

2 Diversified Top- k MaxSAT Problem

2.1 Preliminaries

A literal is either a Boolean variable x (positive literal) or its negation $\neg x$ (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. A formula F in Conjunctive Normal Form (CNF) is a conjunction of clauses, which can be represented as a set of clauses. Any variable in F may take values *true* or *false*. An assignment α_i for F with n variables, x_1, x_2, \dots, x_n , is a mapping that assigns each variable a value and it is expressed by $\alpha_i = v_1 v_2 \dots v_n$, where v_i ($1 \leq i \leq n$) is the corresponding value of x_i . Given an assignment, a clause is satisfied *iff* at least one of its literals is *true*, and a formula F is satisfied *iff* each clause in F is satisfied.

The Maximum Satisfiability (MaxSAT) problem is to find an assignment that satisfies all hard clauses and maximizes the number of satisfied soft clauses of a given CNF formula, $F = \text{Hard} \cup \text{Soft}$, whose clauses can be distinguished into hard clauses and soft clauses, and *Hard* (resp. *Soft*) represents the set of hard (resp. soft) clauses. For a MaxSAT formula F , we say an assignment α is a feasible assignment of F *iff* it satisfies all hard clauses of F . Given a set of feasible assignments, $S = \{\alpha_1, \alpha_2, \dots, \alpha_k\}$, of a MaxSAT formula F , the private satisfied soft clauses of α_i ($\alpha_i \in S$), denoted by $\text{priv}(\alpha_i, S)$, is the subset of soft clauses only satisfied by α_i , i.e., $\text{priv}(\alpha_i, S) = \text{sat}(\alpha_i) \setminus \text{sat}(S \setminus \{\alpha_i\})$, where $\text{sat}(\alpha_i)$ is the set of satisfied soft clauses by α_i .

► **Definition 1** (Diversified Top- k MaxSAT (DTKMS) Problem). *Given a formula $F = \text{Hard} \cup \text{Soft}$ and a positive integer k , the Diversified Top- k MaxSAT problem is to compute a set S with at most k feasible assignments such that these feasible assignments in S satisfy the maximum number of soft clauses of F ; that is to say, $|\text{sat}(\alpha_1) \cup \text{sat}(\alpha_2) \cup \dots \cup \text{sat}(\alpha_k)|$ is maximized, where $\text{sat}(\alpha_i)$ ($\alpha_i \in S$ and $1 \leq i \leq k$) is the set of soft clauses satisfied by α_i .*

It is easy to see that the DTKMS problem is a generalization of MaxSAT, where MaxSAT aims to find one solution, while DTKMS attempts to find k diversified solutions. Since both MaxSAT and DTKMS study the same type of formulae, which involve hard and soft clauses, to avoid confusion, we will distinguish the formulae according to different problems, expressed as DTKMS or MaxSAT formulae. Given a DTKMS formula F and an integer k , we use S to denote the solution of a DTKMS instance F and $\text{sat}(S)$ to denote the set of soft clauses satisfied by S .

► **Definition 2** (Hamming Distance). *Given two feasible assignments α_i and α_j of a DTKMS formula, the hamming distance between α_i and α_j , denoted by $\text{HDist}(\alpha_i, \alpha_j)$, is the number of variables whose corresponding values in the two feasible assignments are different.*

For example, let $\alpha_1 = 110$ and $\alpha_2 = 100$ be two feasible assignments of a formula with three variables x_1, x_2 and x_3 . There is only one variable x_2 with different values in α_1 and α_2 , so the hamming distance between the two feasible assignments is 1. Clearly, for two assignments α_i and α_j , they are not identical if $\text{HDist}(\alpha_i, \alpha_j) > 0$. The higher hamming distance, the higher degree of diversification of two feasible assignments. Note that the hamming distance $\text{HDist}(\alpha_i, \alpha_j)$ can be calculated in polynomial time [30]. Given a set of feasible assignments S and a feasible assignment $\alpha \notin S$, we use $\text{HDist}(\alpha, S) > 0$ to denote α is different from any feasible assignment in S , i.e., for $\forall \alpha_i \in S$, we have $\text{HDist}(\alpha, \alpha_i) > 0$.

■ **Algorithm 1** LS-DTKMS.

Input: a DTKMS instance $F = \text{Hard} \cup \text{Soft}$, an integer k , and a *cutoff* time
Output: a solution of DTKMS S^*

```

1  $m = m_0, S^* = \emptyset;$ 
2 while elapsed time < cutoff do
    //  $m_{max}$  and  $m_0$  are parameters
3     if  $m < m_{max}$  then  $m \leftarrow 2 \times m;$ 
4     else  $m_0 \leftarrow m_0 + 1, m \leftarrow m_0;$ 
5     for each soft clause  $c$  do
6          $\lfloor \text{ClauseConf}[c] = 1$ 
7      $S \leftarrow \text{ConstructS}(F, m);$ 
8     if  $|\text{sat}(S)| = |\text{Soft}|$  then return  $S;$ 
9      $S \leftarrow \text{UpdateS}(F, S, m);$ 
10    if  $|\text{sat}(S)| > |\text{sat}(S^*)|$  then  $S^* \leftarrow S;$ 
11 return  $S^*;$ 

```

2.2 Top- k MaxSAT based DTKMS Algorithm

This subsection presents a top- k MaxSAT based DTKMS algorithm. This algorithm follows a greedy algorithm for the max k -coverage problem, which is the problem of selecting k subsets from a collection of subsets such that their union contains as many elements as possible. Given a DTKMS instance F and an integer k , this algorithm iteratively selects the best feasible assignment that can satisfy the maximum number of soft clauses and then adds it into the solution S until the top- k feasible assignments are added into S . In essence, the whole process can be accomplished by a top- k MaxSAT solver, which outputs the top- k feasible assignments. This algorithm can achieve an approximation ration of 0.632, which is the best-possible polynomial time approximation algorithm for the k -coverage problem [29].

3 A New Local Search Algorithm for DTKMS

This section describes our local search algorithm, called LS-DTKMS, for solving DTKMS on top level. Details of important components will be presented in the following.

3.1 Framework of LS-DTKMS

LS-DTKMS (Algorithm 1) alternatively performs solution construction (*ConstructS*) and solution update (*UpdateS*) until a given cutoff time is reached. At first, LS-DTKMS initializes S^* and m (line 1), where m is a parameter used in the Best From Multiple Selections (BMS) strategy [10]. The BMS strategy is a probabilistic method for choosing a variable of good quality from a large set, which is effective for improving the performance of local search. Specifically, the strategy chooses m random variables, and returns the best one. Then the algorithm enters a loop (lines 2–10). In each iteration, the parameter m is changed to obtain diversified results (lines 3–4), and *ClauseConf* (described in the next subsection) is initialized for each soft clause (lines 5–6). Thereafter a solution S with at most k feasible assignments is constructed (line 7). If all soft clauses are satisfied by S , S is directly returned (line 8); otherwise, *UpdateS* is performed to improve the quality of S (line 9). If the new solution obtained by *UpdateS* is better than the best-found solution S^* , S^* is updated. Finally, when the loop terminates, LS-DTKMS returns S^* (line 11).

3.2 Finding a Single Feasible Assignment with Diversity

Local search for DTKMS requires that the k generated feasible assignments are lowly overlapping with each other so as to enhance the usefulness of each individual feasible assignment. Specifically, when a feasible assignment has been built, DTKMS problem prefers the next feasible assignment that is dissimilar with the old one. Thus, it is vital to generate diversified feasible assignments to be better applied in both *ConstructS* and *UpdateS*. To get such feasible assignments, we define a novel score of variable and a variable selection heuristic.

3.2.1 Score of Variable

The variable scoring function is defined by incorporating the benefit of the number of satisfied soft clauses under the candidate solution S and the increment of total weight of satisfied clauses under the current assignment by flipping a variable. The specific definition is as follows.

► **Definition 3** (score of variable). *Given a DTKMS formula F , a candidate solution S containing at most k assignments, and an assignment $\alpha \in S$, the score of a variable v , denoted by $score(v)$, is defined as $score(v) = \lambda_1 \cdot score_1(v) + \lambda_2 \cdot score_2(v)$, where λ_1, λ_2 ($0 \leq \lambda_1, \lambda_2 \leq 1$) are weighting factors, and $score_1(v)$ and $score_2(v)$ are defined as follows.*

- $score_1(v)$ is defined under the candidate solution S . The $score_1$ of v is $score_1(v) = |sat((S \setminus \{\alpha\}) \cup \{\alpha'\})| - |sat(S)|$, where α' is an assignment obtained by flipping the value of the variable v in α .
- $score_2(v)$ is defined under the current assignment. The $score_2$ of v is defined as $score_2(v) = make(v) - break(v)$, where $make(v)$ and $break(v)$ represent the total weights of the clauses which would become satisfied and falsified under α respectively if the value of v is flipped.

The weights of clauses are set using the clause weighting mechanism [21], which works as follows. At first, the weight of each hard and soft clause is set to 1. Then, the update rules involve: (1) With probability sp , for each satisfied hard clause c with $w(c) > 1$, $w(c) = w(c) - h_inc$, and for each satisfied soft clause c with $w(c) > 1$, $w(c) = w(c) - 1$; (2) With probability $1 - sp$, for each falsified hard clause c , $w(c) = w(c) + h_inc$, and for each falsified soft clause c with $w(c) < weightLimit$, $w(c) = w(c) + 1$. In the above rules, h_inc ($h_inc > 1$) is a constant, and $weightLimit$ is a limit of the maximum weight value of soft clauses.

Apparently, $score_1(v)$ and $score_2(v)$ both encourage the transformation of the clauses from falsified to satisfied. The $score_1(v)$ inclines the contribution of soft clauses from a global perspective. When flipping variables with $score_1(v)$, the variables with the greater increment of the number of satisfied soft clauses under the candidate solution may be chosen. Flipping such variables is beneficial to finding a better solution. The $score_2(v)$ focuses on emphasizing the importance of hard clauses from a local point of view. When flipping variables using $score_2(v)$, it is likely that the picked variables have greater increment of the weight of satisfied hard clauses under the current assignment. Flipping such variables contributes to finding a better assignment. By integrating $score_1(v)$ and $score_2(v)$, our scoring function is more flexible in the sense that it can select variables according to both local and global perspectives.

3.2.2 Variable Selection Heuristic

In this part, two variable selection heuristics are designed to increase the diversity of the search. One is used in initial assignment construction to generate a diversified initial assignment. The other is applied in local search to modify the initial assignment with the aim of finding a feasible assignment.

The first variable heuristic strategy is implemented by using a Boolean array *ClauseConf* to express the state of each soft clause, where $ClauseConf[c] = 1$ identifies the soft clause c is falsified and unselected during each loop iteration in LS-DTKMS; otherwise, $ClauseConf[c] = 0$. We prefer to choose a variable in the clause c with $ClauseConf[c] = 1$. The reason for doing so is that we prefer to choose the falsified soft clauses to satisfy as many soft clauses in $Soft \setminus sat(S)$ as possible. The details of the update rules for *ClauseConf* are given as follows. At first, the *ClauseConf* of each soft clause is initialized to 1. Then *ClauseConf* is updated based on two circumstances.

1. During finding a feasible assignment process: when a soft clause c is selected during one iteration (*ConstructS* and *UpdateS* execution) in LS-DTKMS, then $ClauseConf[c] = 0$.
2. During updating the solution S process: when removing α from S , the *ClauseConf* of each clause c in $priv(\alpha, S)$ is set to 1; when adding α into S , the *ClauseConf* of each clause c in $priv(\alpha, S)$ is set to 0.

After a clause c is picked, a variable is selected randomly from c to flip so as to build a initial assignment dissimilar with the one extracted from the candidate solution S .

The second variable selection heuristic is a two-priority-level heuristic with the purpose of constructing a diversified feasible assignment, which works as follows.

1. The first priority level: If there exist variables whose $score(v) > 0$, choose a variable with BMS strategy, breaking ties by selecting the one that is least recently flipped. In details, if the number of the variables with $score(v) > 0$ is less than a constant m , choose a variable with the greatest $score$; otherwise, a set is built by randomly selecting m variables whose $score(v) > 0$ and then a variable with the highest $score$ in the newly-built set is selected.
2. The second priority level: There are no variables with $score(v) > 0$, which indicates that the local search is stuck in the local optimum. In this case, the weights of the clauses are first updated. Then a random falsified hard clause is selected if such clauses exist; otherwise, a random falsified soft clause is selected. Finally, a variable with the highest $score$ is chosen from the clause, breaking ties by selecting the one that is least recently flipped.

3.2.3 The Framework of Finding a Single Feasible Assignment

The pseudo code of finding a single feasible assignment *FindAssignment* is outlined in Algorithm 2. *FindAssignment* consists of two phases: initial assignment construction phase (lines 2–13) and local search phase (lines 14–25). In the first phase, with a certain probability p (the noise parameter), an assignment is randomly generated (lines 2–4). With a certain probability $1 - p$, the last feasible assignment is extracted from S . Then a variable v is selected using the first variable heuristic strategy and the initial assignment is generated by flipping the selected variable v (lines 6–13). After building an initial assignment α , *FindAssignment* enters the local search phase, which iteratively modifies α until a given time limit is reached. During each iteration, if a new feasible assignment is constructed, which uses the hamming distance to measure, the assignment is returned (lines 15–16). Otherwise, a variable is selected through the two-priority-level heuristic to find a new assignment (lines 17–25). Finally, the assignment α^* is returned (line 26).

Algorithm 2 FindAssignment(F, S, m).

```

1  $\alpha^* = \emptyset$ ;
  // initial assignment construction phase
2 if with probability  $p$  then
3    $\alpha \leftarrow$  generate a random assignment;
4   update the weights of clauses;
5 else
6    $\alpha \leftarrow$  extract the last feasible assignment from  $S$ ;
7    $S \leftarrow S \setminus \alpha$ ;
8   if  $C = \{c \mid \text{ClauseConf}[c] = 1\} \neq \emptyset$  then
9      $c \leftarrow$  randomly select a falsified soft clause under  $\alpha$  in  $C$ ;
10  else
11     $c \leftarrow$  randomly select a falsified soft clause in  $\text{Soft} \setminus \text{sat}(S)$ ;
12   $v \leftarrow$  select a variable from  $c$  with the highest score;
13   $\alpha \leftarrow$  flip  $v$  in  $\alpha$ ;
  // local search phase
14 while elapsed time < cutoff do
15   if  $\alpha$  is a feasible assignment and  $\text{HDist}(\alpha, S) > 0$  then
16      $\alpha^* \leftarrow \alpha$ ; break;
17   if  $D = \{v \mid \text{score}(v) > 0\} \neq \emptyset$  then
18      $v \leftarrow$  select a variable from  $D$  with BMS strategy;
19   else
20     update the weights of clauses;
21     if exists falsified hard clauses then
22        $c \leftarrow$  select a hard clause randomly;
23     else  $c \leftarrow$  select a soft clause randomly;
24      $v \leftarrow$  select a variable from  $c$  with the highest score;
25      $\alpha \leftarrow$  flip  $v$  in  $\alpha$ ;
26 return  $\alpha^*$ ;

```

3.3 Constructing a Candidate Solution for DTKMS Problem

In the subsection, we present the solution construction algorithm *ConstructS* in Algorithm 3 to build a candidate solution S with at most k feasible assignments. In *ConstructS*, if the size of S is less than k or a given time limit is not reached, a loop is executed iteratively (lines 2–6). In each iteration, *FindAssignment* is called to individually generate a diversified feasible assignment (line 4). If the new assignment is not empty, it is inserted into S (line 5). Next, *ClauseConf* is updated according to the update rules (line 6). Note that when a candidate solution has been generated by *ConstructS*, the feasible assignments in S are different, which can be guaranteed by *FindAssignment*.

3.4 Solution Updating for DTKMS Problem

When a candidate solution has been generated by *ConstructS*, our algorithm needs to further improve the quality of the solution, and this leaves a question: how to evaluate the quality of a feasible assignment in a solution. In the following, we define a scoring function on assignments to evaluate the importance of each feasible assignment.

■ **Algorithm 3** ConstructS (F, S, m).

```

1  $S = \emptyset, \alpha = \emptyset;$ 
2 while  $|S| < k$  or elapsed time  $<$  cutoff do
3   if  $|sat(S)| = |Soft|$  then break;
4    $\alpha \leftarrow$  FindAssignment( $F, S, m$ );
5   if  $\alpha \neq \emptyset$  then  $S \leftarrow S \cup \{\alpha\};$ 
6   update ClauseConf;
7 return  $S$ 

```

3.4.1 Score of Assignment

The scoring function on assignments is crucial in the local search algorithm for DTKMS because it provides the measure of each assignment in a solution, which helps to decide how to replace an old assignment from a solution and thus further improve the solution.

► **Definition 4 (score of assignment).** Given a solution S of a DTKMS instance, and an assignment $\alpha \in S$, the score of α is $score(\alpha) = |priv(\alpha, S)|$.

By using the *score of assignment* scheme, a solution updating rule works as follows by given a a solution S of a DTKMS instance, and any an assignment $\alpha \in S$.

- When an assignment needs to be removed from S , we pick the one with the lowest $score(\alpha)$. Note that such an assignment may not be the one with the least number of satisfied soft clauses. Our algorithm LS-DTKMS prefers removing an assignment α_{min} with the lowest $score(\alpha)$ because it does very little contribution to the DTKMS solution, even if it may satisfy a large number of soft clauses.
- After removing an assignment α_{min} with the lowest $score(\alpha)$ from S based on the above method, we construct a new solution S' by adding an assignment α . To guarantee the quality of the solution, the assignment α to be added should hold the Inequality (1).

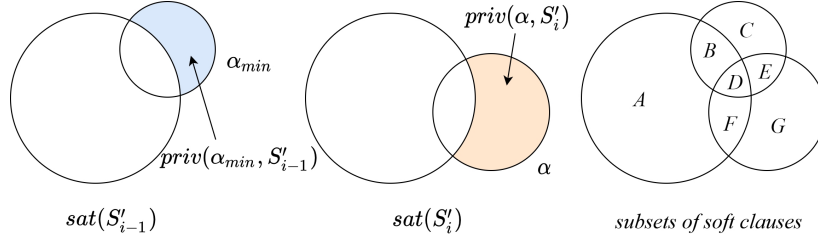
$$|priv(\alpha, S')| > |priv(\alpha_{min}, S)| + \frac{|sat(S)|}{|S|} \quad (1)$$

where $S' = S \setminus \alpha_{min} \cup \{\alpha\}$ and $|S|$ records the number of assignments in the solution S . With the help of the above solution updating rule, we can guarantee that if a solution is obtained according to Inequality (1), then our algorithm LS-DTKMS can achieve a guaranteed approximation ratio of 0.25, which is demonstrated in Lemma 5.

► **Lemma 5.** Let F be a DTKMS instance, k be an integer, S^* be an optimal solution of F , and S' be a solution updated by Inequality (1). Then $|sat(S')| \geq 0.25 \times |sat(S^*)|$.

Proof. When an algorithm for DTKMS constructs a solution S'_i from S'_{i-1} , the condition $|priv(\alpha, S'_i)| > |priv(\alpha_{min}, S'_{i-1})| + \frac{|sat(S'_{i-1})|}{|S'_{i-1}|}$ is equivalent to $|E| + |G| > |C| + |E| + \frac{|sat(S'_{i-1})|}{|S'_{i-1}|}$ represented in Figure 1. Since $|sat(S'_{i-1})| = |A| + |B| + |C| + |D| + |E| + |F|$, $|sat(S'_i)| = |A| + |B| + |D| + |E| + |F| + |G|$, and $|S'_{i-1}| = k$, we have $|sat(S'_i)| > (1 + \frac{1}{k})|sat(S'_{i-1})|$.

Next, we prove the fact that solving the DTKMS problem is equivalent to the max k -coverage. We reduce the max k -coverage problem to the DTKMS problem as follows.



■ **Figure 1** The illustration of Lemma 5.

1. For each element u_i in U , create a variable x_i .
2. For each element u_p in the subset C_i and each element u_q in the subset $U \setminus C_i$, create a hard clause $\neg x_p \vee \neg x_q$.
3. For each element u_i in U , create a soft clause x_i .

We can draw a conclusion that solving the DTKMS problem is equivalent to the max k -coverage problem, which is to select k subsets from $C = \{C_1, C_2, \dots, C_m\}$ such that their union has the maximum cardinality by giving a set U of n elements and an integer k . In addition, we can obtain the Inequality (2) $|cov(C'_i)| > (1 + \frac{1}{k})|cov(C'_{i-1})|$, where $cov(C'_i)$ is the set of elements covered by a solution C'_i of a max k -coverage instance. When Inequality (2) is satisfied, a theoretical result that a max k -coverage algorithm can achieve a guarantee approximation ration of 0.25 is proved in [5]. Therefore, we can conclude that $|sat(S')| \geq 0.25 \times |sat(S^*)|$. ◀

3.4.2 The Framework of Solution Updating

The framework of solution updating algorithm *UpdateS* is presented in Algorithm 4. At first, *UpdateS* initializes the solution S' and *step* (line 1). Then it enters a loop (lines 2–10), in which *UpdateS* iteratively generates a feasible assignment and updates S with the solution updating rule described in the above. Significantly, if a solution is obtained by Inequality (1), our algorithm can achieve a guaranteed approximation ratio of 0.25. Finally, a better solution is returned (line 11).

■ **Algorithm 4** UpdateS (F, S, m).

```

1 step = 0,  $S' = \emptyset$ ;
2 while within the time limit do
3   if step > maxstep then break;
4    $\alpha \leftarrow \text{FindAssignment}(F, S, m)$ ;
5    $\alpha_{min} \leftarrow$  select an assignment  $\alpha$  with the lowest score( $\alpha$ ) from  $S$ ;
6    $S' = S \setminus \alpha_{min} \cup \{\alpha\}$ ;
7   if  $|priv(\alpha, S')| > |priv(\alpha_{min}, S)| + \frac{|sat(S)|}{|S|}$  then
8      $S \leftarrow S', \textit{step} \leftarrow 0$ ;
9   else step  $\leftarrow \textit{step} + 1$ ;
10  update ClauseConf;
11 return  $S$ ;

```

■ **Table 1** Description of the top- k track benchmarks in MSE 2020.

Families	Var	Hard	Soft	Families	Var	Hard	Soft
aes (1)	147	240	147	maxone (2)	485	2817	485
aes-key-recovery (1)	21368	372240	407	MaxSATQueriesinInterpretableClassifiers (5)	1991	14448	1058
atcoss (2)	192216	890542	301	mbd (2)	29703	73188	4637
bcp (2)	152	462	125	packup (5)	13138	73769	7572
CircuitDebuggingProblems (2)	144580	0	432737	protein_ins (5)	2253	2190700	59
CircuitTraceCompaction (2)	158890	494821	20	pseudoBoolean (3)	839	1610	815
close_solutions (4)	68053	2600438	67868	railway-transport (2)	64743	1668566	4945
ConsistentQueryAnswering (3)	44526	46593	10856	ramsey (1)	36	0	210
des (3)	86878	388071	4707	reversi (6)	2981	16521	45
drmx-atmostk (3)	927	1408	47	scheduling (1)	201204	767081	1707
fault-diagnosis (1)	137900	758138	49770	SeanSafarpour (1)	238290	0	936006
frb (1)	760	41367	760	treewidth-computation (3)	88809	814018	60
gen-hyper-tw (1)	62531	197071	48	uaq (3)	2205	3805	189
maxelique (3)	182	16885	182	xai-mindset2 (1)	1330	4763	378
MaximumCommonSub-GraphExtraction (3)	2044	99615	41				

4 Experimental Evaluation

In this section, we carry out two experiments to evaluate the performance of LS-DTKMS. The first experiment compares LS-DTKMS against top- k MaxSAT based DTKMS algorithms on top- k maxsat instances from the MaxSAT Evaluation (MSE) 2020 top- k track². The second experiment applies LS-DTKMS to Diversified Top- k Clique Search (DTKCS) problem, whose instances are from DIMACS graph benchmarks³.

4.1 Experimental Preliminaries

Our algorithm LS-DTKMS is implemented in C++ and compiled by g++ with “-o3”. There are 9 parameters in it. (1) m_0, m_{max} : the initial and the maximum value of the number of samplings used in BMS, respectively. (2) p : the probability, used to generate an initial assignment. (3) sp : the smooth probability, used in the clause weighting scheme. (4) h_{inc} : the increment of falsified hard clauses, used in the clause weighting scheme. (5) *WeightLimit*: the limit on soft clause weight, used in the clause weighting scheme. (6) *maxstep*: the limit on the step of updating the solution. (7) λ_1, λ_2 : two coefficients of $score(v)$, used for variable selection. The parameters are tuned according to our experience, and are listed as follows: $m_0 = 15$, $m_{max} = 125$, $p = 0.2$, $sp = 0.01$, $h_{inc} = 2$, $weightLimit = 1$, $maxstep = 23 \times 10^6$, $\lambda_1 = 0.6$, and $\lambda_2 = 0.4$.

The first experiment compares LS-DTKMS with four top- k MaxSAT based DTKMS algorithms, all of which exploit top- k MaxSAT solvers from top- k track in MSE 2020. The solvers in the top- k track can return k best feasible assignments, which follows the idea of the DTKMS algorithm based on top- k MaxSAT. In the experiment, we use four types of top- k MaxSAT solvers: MaxHS [6], Open-WBO [26], Maxino[3], and RC2 [18]. In each type, we exploit the best one, namely, MaxHS, Open-WBO, maxino, and RC2-A, where RC2-A is the best top- k MaxSAT solver by far. In the second experiment, we evaluate LS-DTKMS with two state-of-the-art incomplete DTKCS solvers, TOPKLS [32] and HEA-D [34]. TOPKLS and HEA-D are state-of-the-art incomplete solvers for DTKCS problem, where HEA-D is the best one by far. In addition, to ensure LS-DTKMS can solve DTKCS benchmarks, we model these instances into DTKMS instances using usual encoding [22]. All experiments are conducted on a server with an Intel(R) Xeon(R) 2.10GHz CPU and 256GB of memory under CentOS Linux release 7.9.2009. For each instance, four top- k MaxSAT based DTKMS

² <https://maxsat-evaluations.github.io/2020/benchmarks.html>

³ https://iridia.ulb.ac.be/~fmascia/maximum_clique/

■ **Table 2** Comparison on top- k maxsat with $k=2$ and $k=3$.

Families	$k=2$					$k=3$				
	Open-WBO	MaxHS	Maxino	RC2-A	LS-DTKMS	Open-WBO	MaxHS	Maxino	RC2-A	LS-DTKMS
aes (1)	NA(600.00)	129(25.34)	130(0.23)	130(0.20)	147(7.38)	NA(600.00)	132(25.34)	132(0.23)	132(0.19)	147(8.37)
aes-key-recovery (1)	NA(600.00)	406(600.00)	NA(600.00)	406(600.00)	407(40.92)	NA(600.00)	406(600.00)	NA(600.00)	406(600.00)	407(62.34)
atcoss (2)	265.5(364.80)	265.5(23.54)	265.5(25.87)	265.5(43.51)	290.5(53.49)	265.5(102.71)	265.5(23.96)	265.5(26.24)	265.5(43.83)	290.5(56.31)
bcp (2)	73.5(0.02)	73.5(0.01)	73.5(0.01)	73.5(0.00)	74(7.43)	73.5(0.02)	73.5(0.01)	73(0.01)	73.5(0.01)	74(7.93)
CircuitDebugging-Problems (2)	432736	432736	432736	432736	432737	432736	432736.5	432736	432736	432736.5
CircuitTraceCo-mpaction (2)	12.5(22.60)	12(37.54)	12.5(9.52)	12.5(20.99)	14(13.61)	13(22.02)	13(38.90)	13(9.58)	13(20.28)	16.5(11.55)
close_solutions (4)	67857	67857	67857.3	67857	67861.3	67858.3	67858.3	67858.3	67858.3	67862
ConsistentQuery-Answering (3)	(44.56)	(178.48)	(12.45)	(2.45)	(28.55)	(43.71)	(158.82)	(12.46)	(2.92)	(31.66)
des (3)	0(1.63)	0(0.05)	0(0.25)	0(0.05)	0(21.42)	0(1.62)	0(0.06)	0(0.25)	0(0.06)	0(23.06)
drmx-atmostk (3)	3978(251.09)	3977(333.09)	4699.3(112.24)	4700(30.56)	4701.7(103.27)	3978(251.61)	3978(251.17)	4699.3(112.79)	4700(30.25)	4701.7(61.81)
fault-diagnosis (1)	29(0.67)	29(98.53)	19(201.58)	28.7(21.05)	32(132.99)	29.3(0.68)	30(94.93)	19(201.60)	29.7(21.33)	44(125.54)
frb (1)	49593	49593	49593	49593	49593	49593	49593	49593	49593	49593
gen-hyper-tw (1)	(34.63)	(122.39)	(24.46)	(7.01)	(8.22)	(38.30)	(150.28)	(25.17)	(7.39)	(8.43)
MaximumCommonSub-Graph-Extraction (3)	40(46.34)	NA(600.00)	NA(600.00)	42(6.97)	54(84.00)	40(46.56)	NA(600.00)	NA(600.00)	43(7.14)	77(89.27)
maxclique (3)	44(128.45)	44(117.65)	44(233.96)	44(117.94)	44(121.98)	44(131.84)	44(61.67)	44(242.70)	44(116.78)	44(127.47)
MaxSATQuery-interpretation-Classifiers (5)	18.7(1.37)	20(56.54)	19.7(4.91)	19.3(0.56)	30(11.58)	116.3(1.39)	116(61.49)	116.3(4.96)	116.3(0.00)	135.7(10.55)
mbd (2)	19(226.61)	20.3(130.08)	20.3(122.02)	20(107.98)	22.3(149.94)	20(224.80)	21.3(218.50)	21.3(120.73)	21(105.84)	25(152.61)
packup (5)	238.5(0.18)	238(0.18)	238.5(0.05)	238(0.21)	254(79.91)	245(0.18)	240(0.18)	240(0.05)	238(0.23)	257.5(73.90)
protein_ins (5)	527.2(23.11)	553.3(89.70)	525.6(1.98)	526.6(6.21)	556(68.68)	528.2(23.48)	554.5(120.08)	526.6(1.98)	528.2(6.62)	557.2(69.06)
pseudoBoolean (3)	4616(198.04)	4616(61.28)	4615.5(1.70)	4616(0.59)	4636(0.17)	4617(194.40)	4617(58.63)	4616.5(1.70)	4617(0.61)	4636(0.19)
ramsey (1)	6959.8(11.81)	6957.4(6.58)	6958.2(1.03)	6959.2(0.78)	7209.8(25.10)	6965.4(11.98)	6966.6(6.88)	6966(1.03)	6964.8(0.91)	7209.8(29.17)
reversi (6)	29.6(85.07)	30.4(98.79)	30(206.86)	29.6(33.40)	38.2(140.30)	30.6(88.13)	31(192.14)	30.6(188.86)	29.8(33.89)	39.8(104.50)
seanSafarpour (1)	408(0.81)	408(0.05)	407.7(0.07)	408(0.04)	409(1.70)	408(0.78)	408.3(0.05)	407.7(0.07)	408(0.05)	410(2.23)
tree-width-computation (3)	4920.5(27.41)	4921(44.00)	4920.5(6.67)	4920.5(4.36)	4932.5(14.94)	4775.5(27.95)	4775.5(34.61)	4775.5(6.71)	4775.5(4.43)	4932.5(19.35)
uaq (3)	210(0.10)	210(0.65)	210(0.72)	210(0.31)	210(16.27)	210(0.10)	210(0.66)	210(0.69)	210(0.32)	210(16.32)
xai-mindset2 (1)	32.3(2.01)	31.8(10.50)	31.8(6.52)	32.17(4.97)	37.3(46.55)	33(2.08)	32.7(9.97)	32.5(6.55)	32.8(5.26)	39.3(47.57)
	1478(600.00)	1476(209.98)	1478(29.94)	1478(107.71)	1481(297.34)	1478(600.00)	1477(600.00)	1478(29.94)	1478(104.15)	1481(162.31)
	NA(600.00)	NA(600.00)	936004	936004	NA(600.00)	NA(600.00)	NA(600.00)	936004	936004	NA(600.00)
			(312.62)	(44.42)				(320.09)	(48.05)	
	52.8(28.90)	52.8(26.56)	52.8(16.11)	52.8(15.32)	52.8(17.99)	52.8(37.75)	52.8(33.12)	52.8(16.09)	52.8(15.25)	52.8(20.22)
	140.3(10.06)	136.7(16.81)	137.3(25.72)	141.3(7.88)	170(79.88)	140.3(10.08)	138.3(17.95)	138.3(26.89)	149.3(7.97)	173(84.54)
	374(0.07)	372(0.06)	374(0.02)	359(0.02)	376(22.43)	374(0.07)	372(0.07)	374(0.03)	360(0.01)	376(27.31)

solvers are executed once, and all other incomplete solvers, including LS-DTKMS, TOPKLS, and HEA-D are executed 10 times with different settings of random seeds. The cutoff time for them is set to 600 seconds. Note that we use the default values of all the parameters for TOPKLS, HEA-D, Open-WBO, Maxino, RC2-A, and MaxHS.

4.2 Experiment Results on Top-k MaxSAT

Table 2 and 3 show the results of the comparison of LS-DTKMS with four top- k MaxSAT based DTKMS solvers on 73 top- k MaxSAT instances (belong to 29 families). Since for some instances when k is set to 5 or more, all soft clauses are satisfied, we set the parameter k to 2, 3, 4, and 5, respectively. In the two tables, the first column records the name of each family as well as the number of instances in each family (in brackets). For each family of instances, we report the average number of variables ($|Var|$), hard clauses ($|Hard|$), and soft clauses ($|Soft|$) (Table 1), the average number of satisfied soft clauses and the average runtime in seconds (in brackets) that each solver can solve within the cutoff time (Table 2 and 3). If a solver fails to find a feasible solution, the corresponding result is marked with “NA”. As shown in the two tables, LS-DTKMS achieves the best results in terms of solution quality on most families. In addition, as k gets larger, the solution quality of LS-DTKMS are almost stable. Hence, the performance of LS-DTKMS is considerably better than the ones of four top- k MaxSAT based DTKMS solvers. Moreover, the runtime of LS-DTKMS is comparable to the other solvers for the majority of instances.

■ **Table 3** Comparison on top- k maxsat with $k=4$ and $k=5$.

Families (29)	$k=4$				$k=5$					
	Open-WBO	MaxHS	Maxino	RC2-A	LS-DTKMS	Open-WBO	MaxHS	Maxino	RC2-A	LS-DTKMS
aes (1)	NA(600.00)	132(25.53)	132(0.23)	132(0.20)	147(6.12)	NA(600.00)	132(25.95)	132(0.23)	132(0.21)	147(5.12)
aes-key-recovery (1)	NA(600.00)	406(600.00)	NA(600.00)	406(600.00)	407(73.34)	NA(600.00)	406(600.00)	NA(600.00)	406(600.00)	407(75.12)
atcross (2)	265.5(108.59)	265.5(25.35)	265.5(22.44)	265.5(53.06)	290.5(57.96)	265.5(110.47)	265.5(26.79)	265.5(22.90)	265.5(57.42)	290.5(62.12)
bcp (2)	73.5(0.01)	73.5(0.01)	73(0.01)	73.5(0.00)	74(7.78)	73.5(0.01)	73.5(0.01)	73(0.01)	73.5(0.01)	74(6.99)
CircuitDebugging-Problems (2)	432736(321.45)	432736.5(299.95)	432736(9.11)	432736(9.11)	432736.5(23.08)	432736.5(322.19)	432736.5(300.17)	432736.5(6.04)	432736.5(10.00)	432736.5(24.57)
CircuitTraceCompaction (2)	13(20.56)	13(26.65)	13(6.31)	13(26.10)	18.5(14.03)	13(21.41)	13(30.88)	13(6.25)	13(26.68)	20(14.28)
close_solutions (4)	67858.3(45.49)	67858.3(87.84)	67858.3(17.21)	67858.3(4.35)	67866(37.92)	67858.3(44.75)	67858.3(130.86)	67858.3(17.03)	67858.3(4.81)	67866(40.90)
ConsistentQuery-Answering (3)	0(1.63)	0(0.05)	0(0.24)	0(0.04)	0(21.38)	0(1.65)	0(0.05)	0(0.25)	0(0.05)	0(22.46)
des (3)	3978.5(246.31)	3978.5(329.68)	4700(122.37)	4700.7(31.35)	4701.7(58.56)	4706.5(244.65)	4706.5(213.85)	4625.3(110.95)	4626(55.35)	4626.3(113.76)
drmx-atmostk (3)	31(0.67)	31.7(99.74)	22(201.60)	31.3(21.58)	46.7(116.70)	31(0.66)	31.7(96.77)	22(201.63)	31.7(21.65)	46.7(116.83)
fault-diagnosis (1)	49593(30.24)	49593(150.27)	49593(20.74)	49593(7.59)	49593(11.28)	49593(30.27)	49593(180.01)	49593(20.25)	49593(7.69)	49593(8.12)
frb (1)	45(43.69)	NA(600.00)	NA(600.00)	45(7.00)	110(99.09)	45(44.15)	NA(600.00)	NA(600.00)	45(7.04)	122(123.14)
gen-hyper-tw (1)	44(178.53)	44(105.23)	44(221.92)	44(116.71)	38(88.79)	44(181.03)	44(188.43)	44(223.14)	44(112.24)	42(94.13)
maxclique (3)	117(2.03)	116.7(60.98)	117(5.03)	117(0.01)	143(11.09)	117(1.50)	116.7(138.22)	117(5.00)	117(0.00)	150.3(10.49)
MaximumCommonSub-Graph-Extraction (3)	23(219.66)	22.3(187.72)	23(107.63)	23(104.37)	26(151.78)	23(224.80)	22.3(197.61)	23(107.63)	23.7(105.67)	27(149.65)
maxone (2)	245(0.23)	243(0.18)	245(0.05)	238(0.22)	258(69.85)	248(0.17)	245(0.19)	245.5(0.03)	238(0.23)	258(64.77)
MaxSATQueryInterpretation-Classifiers (5)	528.2(22.02)	554.5(120.58)	526.6(1.84)	528.2(6.87)	557.2(67.97)	528.2(23.08)	554.5(110.08)	526.6(1.87)	528.2(6.93)	557.2(64.76)
mbd (2)	4617(259.82)	4617(42.93)	4616.5(1.85)	4617.5(0.69)	4636(0.29)	4617.5(255.90)	4617(47.07)	4617(1.83)	4618(0.67)	4636(0.16)
packup (5)	6972.4(11.34)	6972.4(7.04)	6973.2(0.99)	6971.2(0.89)	7209.8(28.74)	6972.6(11.38)	6972.6(7.32)	6973.4(0.98)	6973.2(0.92)	7209.8(26.52)
protein_ins (5)	31.6(91.45)	31.8(180.38)	31.6(209.63)	30.4(34.42)	48(104.57)	33.6(102.35)	33.6(174.08)	33.8(167.46)	31.4(35.70)	55.4(101.52)
pseudoBoolean (3)	408(0.82)	408.3(0.05)	408.3(0.07)	408(0.05)	410(1.96)	408(0.83)	408.3(0.05)	408.3(0.06)	408(0.05)	410(0.91)
railway-transport (2)	4777(21.63)	4777(38.16)	4777(7.38)	4777(4.30)	4932.5(23.28)	4777(27.77)	4777(38.07)	4777(7.28)	4777(4.79)	4932.5(24.23)
ramsey (1)	210(0.10)	210(0.70)	210(0.72)	210(0.30)	210(14.12)	210(0.10)	210(0.81)	210(0.32)	210(0.32)	210(10.31)
reversi (6)	33.8(2.08)	33.3(9.71)	33.3(6.92)	33.8(4.65)	40.7(48.64)	33.8(2.21)	33.3(19.36)	33.3(6.98)	33.8(5.12)	41.3(42.49)
scheduling (1)	1481(600.00)	1479(600.00)	1481(37.57)	1479(98.38)	1481(183.23)	1705(18.41)	1705(416.33)	1705(37.95)	1705(29.76)	1707(18.01)
SeanSafarpour (1)	NA(600.00)	NA(600.00)	936004(315.25)	936004(45.29)	NA(600.00)	NA(600.00)	NA(600.00)	936004(335.92)	936004(56.57)	NA(600.00)
treewidth-computation (4)	52.8(41.45)	52.8(32.04)	52.8(14.79)	52.8(14.38)	52.8(18.57)	52.8(45.45)	52.8(31.81)	52.8(14.75)	52.8(15.61)	52.8(26.67)
uaq (3)	140.3(15.97)	138.3(20.87)	138.3(29.22)	155(10.00)	183(82.52)	140.3(16.46)	138.3(27.23)	138.3(31.06)	155(11.57)	183(73.48)
xai-mindset2 (1)	374(0.07)	372(0.08)	374(0.03)	361(0.00)	376(29.21)	374(0.07)	372(0.11)	374(0.03)	362(0.03)	376(45.13)

4.3 Experiment Results on DTKCS

In this subsection, we compare LS-DTKMS against two solvers on 37 DIMACS instances and set the parameter k to 5, 10, 15, and 20, respectively. To save space, we omit the experiment results that cannot be encoded into DTKMS and only display the ones of the 21 instances in Table 3 and 4. In the two tables, we report the number of hard clauses ($|hard|$), the number of soft clauses ($|soft|$), the largest number of satisfied soft clauses ($best$), the average one (avg), and the average time in seconds obtained by executing each solver ten times ($time$). Note that the scale of each instance is listed according to the transformed DIMACS instance using usual encoding, which causes the number of variables is equal to the number of soft clauses. The results show that the performance of LS-DTKMS is surprisingly good on most of instances. LS-DTKMS outperforms the other two solvers on all instances when $k=15$, and on most instances when $k=5, 10$ and 20 . Specifically, LS-DTKMS cannot find the best solution for just 1 instance (C125.9 when $k=10$), which demonstrates that LS-DTKMS is a competitive DTKMS solver.

5 Related work

In this section, we review the related work, including the diversity in SAT and other diversified top- k problems.

Diversity in SAT. The diversity has been studied in the SAT problem and its related problems. For example, Agbaria et al. studied the diversity in SAT, whose diversity is measured by the value of the average distance between each pair of solutions normalized by the number of variables. They proposed two SAT-based methods to generate diverse

■ **Table 4** Comparison on DTKCS with $k=5$ and $k=10$.

Instances (21)	Hard	Soft	$k=5$								$k=10$							
			TOPKLS		HEA-D		LS-DTKMS		TOPKLS		HEA-D		LS-DTKMS					
			<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>				
brock200_2	10024	200	52(52)	110.63	52(52)	151.98	52(52)	53.96	93(93)	126.26	91(90)	210.78	99(99)	211.32				
brock200_4	6811	200	73(73)	378.36	75(74)	169.93	77(75)	128.59	121(121)	36.53	133(130)	471.57	135(133)	35.17				
brock400_2	20014	400	110(110)	187.87	115(112)	293.57	120(120)	207.95	192(192)	25.51	191(190)	263.77	229(228)	339.11				
brock400_4	20035	400	119(119)	159.26	122(118)	352.24	129(129)	265	192(192)	530.14	191(190)	196.11	231(231)	225.13				
C1000.9	49421	1000	271(270)	292.07	264(259)	534.38	321(319)	281.39	464(464)	497.64	474(471)	219.83	507(507)	116.46				
C125.9	787	125	101(101)	73.04	113(112)	6.90	123(122)	5.89	104(104)	0.03	125(125)	0.00	122(120)	9.62				
C250.9	3141	250	155(155)	553.33	173(167)	154.54	193(193)	28.48	206(206)	406.89	250(249)	0.02	250(246)	2.32				
C500.9	12418	500	212(212)	222.91	224(219)	372.79	257(256)	147.64	323(321)	285.76	372(369)	329.21	443(441)	284.39				
gen200_p0.9_44	1990	200	130(130)	561.09	149(145)	55.35	174(174)	223.79	166(166)	249.74	200(200)	0.00	200(197)	15.34				
gen200_p0.9_55	1990	200	145(145)	392.94	160(157)	5.93	185(185)	106.08	174(174)	84.37	200(200)	0.00	200(200)	70.21				
gen400_p0.9_55	7980	400	184(184)	193.76	200(197)	296.86	219(207)	209.95	264(264)	127.53	334(329)	179.58	380(380)	351.47				
gen400_p0.9_65	7980	400	189(189)	283.49	217(210)	283.67	250(250)	218.55	274(274)	453.39	340(338)	262.97	377(377)	191.76				
gen400_p0.9_75	7980	400	196(196)	402.29	240(232)	192.26	265(265)	220.12	276(276)	74.51	344(342)	113.96	367(367)	57.04				
hamming8-4	11776	256	80(80)	10.05	80(80)	4.54	80(80)	8.33	160(160)	0.98	160(160)	75.85	160(160)	8.2				
keller4	5100	171	55(55)	7.29	55(55)	11.01	55(55)	2.24	99(99)	274.42	98(97)	248.39	110(110)	114.16				
MANN_a27	702	378	366(366)	524.8	378(378)	0.00	378(375)	260.1	378(378)	100.54	378(378)	0.00	378(378)	52.15				
MANN_a45	1980	1035	973(973)	462.08	1035(1035)	0.01	1035(1032)	35.85	1034(1034)	62.14	1035(1035)	0.01	1035(1035)	66.36				
MANN_a81	6480	3321	3036(3032)	335.67	3321(3321)	0.12	3321(3320)	31.47	3301(3301)	88.8	3321(3321)	0.12	3321(3321)	1.52				
p_hat300-1	33917	300	39(39)	7.67	39(39)	55.04	39(39)	7.05	73(73)	85.72	68(68)	226.72	74(73)	268.54				
p_hat300-2	22922	300	79(79)	222.49	80(77)	111.64	94(93)	55.15	110(110)	73.2	130(127)	373.91	150(150)	343.28				
p_hat300-3	11460	300	108(108)	9.00	116(113)	232.32	138(138)	6.69	148(147)	251.25	193(187)	408.68	195(195)	109.7				

solutions (satisfying assignments) of SAT for use in the hardware semiformal verification [1]. As a detailed extension of [1], Nadel further discussed the diverse k Set problem in SAT, that is, the problem of efficiently generating a number of diverse solutions given a formula [28]. Alòs et al. proposed a minimum decision tree computation algorithm based on MaxSAT encoding, where one of the tasks is to generate diverse solutions with different variable assignments to extract multiple minimum decision trees. The diversity is enforced by target variables during the incremental calls to the SAT solver, allowing the algorithm to favour the polarity of target variables that were less frequent in previous solutions [2].

Diversified top- k problem. The diversified top- k problem has been extensively studied, which aims to find diversified top- k results. [16, 32, 34, 37] studied the diversified top- k clique problem and [33] worked on the diversified top- k s -plex problem, both of which are to find k cliques or s -plex to maximize the number of covered vertices. Fan et al. studied the diversified top- k graph pattern matching problem, which to find a set of k matches such that the bi-criteria diversification function is maximized [14]. Liu et al. defined the k shortest paths with diversity problem of finding top- k shortest paths to minimize the total length [24]. Xu et al. proposed two exact algorithms for the spatial diversified top- k routes (SDkR) query to obtain k trip routes with high popularity [36]. Lin et al. introduced the diversified top- k lasting cohesive subgraphs problem, which finds k maximal lasting (k, σ)-cores with maximum coverage regarding the number of vertices and timestamps [23]. Lyu et al. presented an algorithm for the diversified top- k biclique search problem which aims to find k maximal bicliques that cover the maximum number of edges [25]. Overall, the diversity top- k problem is becoming increasingly important research field.

6 Conclusion and Future Work

In this study, we propose a local search algorithm for DTKMS, called LS-DTKMS, which features scoring functions to select variables and assignments. The results show that LS-DTKMS achieves good performance across a broad range of instances, including the top- k MaxSAT instances and DTKCS instances. In the future, we will attempt to further improve LS-DTKMS via a few novel heuristic rules.

■ **Table 5** Comparison on DTKCS with $k=15$ and $k=20$.

Instances (21)	Hard	Soft	$k=15$						$k=20$					
			TOPKLS		HEA-D		LS-DTKMS		TOPKLS		HEA-D		LS-DTKMS	
			<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>	<i>best(avg)</i>	<i>time</i>
brock200_2	10024	200	125(125)	296.52	128(126)	230.60	143(143)	70.22	146(146)	296.52	155(154)	372.79	164(160)	339.04
brock200_4	6811	200	151(151)	20.98	167(163)	410.85	191(190)	25.73	170(170)	20.98	192(188)	94.76	198(196)	44.24
brock400_2	20014	400	248(248)	491.45	264(263)	231.79	320(320)	220.73	292(292)	491.45	319(318)	240.85	369(369)	259.81
brock400_4	20035	400	252(252)	142.27	263(262)	305.84	327(327)	256.96	287(287)	142.27	325(318)	184.80	360(360)	180.30
C1000.9	49421	1000	587(587)	177.08	664(657)	136.11	770(770)	316.18	689(689)	177.08	841(833)	56.72	823(823)	372.70
C125.9	787	125	125(125)	0.00	125(125)	0.00	125(125)	19.74	125(125)	0.00	125(125)	0.00	125(125)	11.92
C250.9	3141	250	226(226)	86.74	250(250)	0.00	250(250)	119.19	223(223)	86.74	250(250)	0.00	250(250)	4.16
C500.9	12418	500	383(383)	57.54	490(488)	27.51	500(500)	142.67	418(418)	57.54	500(500)	0.00	500(500)	64.40
gen200_p0.9_44	1990	200	168(168)	2.83	200(200)	0.00	200(200)	9.16	200(200)	2.83	200(200)	0.00	200(200)	18.28
gen200_p0.9_55	1990	200	172(172)	1.61	200(200)	0.00	200(200)	11.77	200(200)	1.61	200(200)	0.00	200(200)	32.98
gen400_p0.9_55	7980	400	310(310)	43.22	400(400)	0.03	400(400)	9.42	326(325)	43.22	400(400)	0.01	400(400)	44.03
gen400_p0.9_65	7980	400	319(319)	88.31	400(400)	0.02	400(400)	12.81	341(341)	88.31	400(400)	0.01	400(400)	39.76
gen400_p0.9_75	7980	400	311(311)	442.95	400(400)	0.01	400(400)	11.35	335(335)	442.95	400(400)	0.00	400(400)	69.26
hamming8-4	11776	256	224(224)	69.23	240(229)	416.12	240(240)	41.98	246(246)	69.23	222(220)	353.77	251(251)	318.61
keller4	5100	171	126(126)	535.79	131(130)	218.73	149(149)	206.85	142(142)	535.79	152(151)	171.54	153(151)	52.11
MANN_a27	702	378	378(378)	0.14	378(378)	0.00	378(378)	6.13	378(378)	0.00	378(378)	0.00	378(378)	4.64
MANN_a45	1980	1035	1035(1033)	1.55	1035(1035)	0.01	1035(1035)	13.31	1035(1035)	1.55	1035(1035)	0.01	1035(1035)	57.91
MANN_a81	6480	3321	3321(3321)	338.63	3321(3321)	0.13	3321(3321)	116.70	3321(3321)	338.63	3321(3321)	0.13	3321(3321)	44.86
p_hat300-1	33917	300	108(105)	360.05	98(97)	198.69	108(107)	251.24	125(125)	360.05	121(118)	229.45	140(140)	224.28
p_hat300-2	22922	300	134(134)	57.61	162(159)	344.25	191(191)	257.23	144(144)	57.61	187(185)	194.29	220(219)	44.66
p_hat300-3	11460	300	168(168)	54.77	240(234)	156.88	276(276)	233.52	177(177)	54.77	275(274)	207.40	276(272)	46.18

References

- 1 Sabih Agbaria, Dan Carmi, Orly Cohen, Dmitry Korchemny, Michael Lifshits, and Alexander Nadel. Sat-based semiformal verification of hardware. In Roderick Bloem and Natasha Sharygina, editors, *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23*, pages 25–32. IEEE, 2010. doi:10.5555/1998496.1998505.
- 2 Josep Alòs, Carlos Ansótegui, and Eduard Torres. Interpretable decision trees through MaxSAT. *Artificial Intelligence Review*, pages 1–21, 2022. doi:10.1007/s10462-022-10377-0.
- 3 Mario Alviano. Maxino. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 21–22, 2020.
- 4 Josep Argelich, Chu Min Li, Felip Manyà, and Zhu Zhu. MinSAT versus MaxSAT for optimization problems. In Christian Schulte, editor, *Principles and Practice of Constraint Programming – 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 133–142. Springer, 2013. doi:10.1007/978-3-642-40627-0_13.
- 5 Giorgio Ausiello, Nicolas Boria, Aristotelis Giannakos, Giorgio Lucarelli, and Vangelis Th. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14):1901–1913, 2012. doi:10.1016/j.dam.2012.04.005.
- 6 Fahiem Bacchus. MaxHS in the 2020 MaxSAT evaluation. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 19–20, 2020.
- 7 Hamid Ahmadi Beni and Asgarali Bouyer. TI-SC: top-k influential nodes selection based on community detection and scoring criteria in social networks. *Journal of Ambient Intelligence and Humanized Computing*, 11(11):4889–4908, 2020. doi:10.1007/s12652-020-01760-2.
- 8 Jeremias Berg and Matti Järvisalo. Optimal correlation clustering via MaxSAT. In Wei Ding, Takashi Washio, Hui Xiong, George Karypis, Bhavani Thuraisingham, Diane J. Cook, and Xindong Wu, editors, *13th IEEE International Conference on Data Mining Workshops, ICDM Workshops, TX, USA, December 7-10, 2013*, pages 750–757. IEEE Computer Society, 2013. doi:10.1109/ICDMW.2013.99.
- 9 Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, António Morgado, and João Marques-Silva. Propositional proof systems based on maximum satisfiability. *Artificial Intelligence*, 300:103552, 2021. doi:10.1016/j.artint.2021.103552.

- 10 Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 747–753. AAAI Press, 2015. URL: <http://ijcai.org/Abstract/15/111>.
- 11 Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020. doi:10.1016/j.artint.2020.103354.
- 12 Mohamed Sami Cherif, Djamal Habet, and André Abramé. Understanding the power of max-sat resolution through up-resilience. *Artificial Intelligence*, 289:103397, 2020. doi:10.1016/j.artint.2020.103397.
- 13 Rafael Dutra, Kevin Laeuffer, Jonathan Bachrach, and Koushik Sen. Efficient sampling of SAT solutions for testing. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 – June 03, 2018*, pages 549–559. ACM, 2018. doi:10.1145/3180155.3180248.
- 14 Wenfei Fan, Xin Wang, and Yinghui Wu. Diversified top-k graph pattern matching. *Proceedings of the VLDB Endowment*, 6(13):1510–1521, 2013. doi:10.14778/2536258.2536263.
- 15 Kevin Gimpel, Dhruv Batra, Chris Dyer, and Gregory Shakhnarovich. A systematic exploration of diversity in machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1100–1111. ACL, 2013. URL: <https://aclanthology.org/D13-1111/>.
- 16 Fei Hao, Zheng Pei, and Laurence T. Yang. Diversified top-k maximal clique detection in Social Internet of Things. *Future Generation Computer Systems*, 107:408–417, 2020. doi:10.1016/j.future.2020.02.023.
- 17 Hao Hu, Marie-José Huguet, and Mohamed Siala. Optimizing binary decision diagrams with MaxSAT for classification. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*, pages 3767–3775. AAAI Press, 2022. doi:10.1609/aaai.v36i4.20291.
- 18 Alexey Ignatiev. RC2-2018@ MaxSAT evaluation 2020. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 13–14, 2020.
- 19 Alexey Ignatiev, Yacine Izza, Peter J. Stuckey, and João Marques-Silva. Using MaxSAT for efficient explanations of tree ensembles. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI 2022*, pages 3776–3785. AAAI Press, 2022. doi:10.1609/aaai.v36i4.20292.
- 20 Mohit Kumar, Samuel Kolb, Stefano Teso, and Luc De Raedt. Learning MAX-SAT from contextual examples for combinatorial optimisation. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 4493–4500. AAAI Press, 2020. doi:10.1609/aaai.v34i04.5877.
- 21 Zhendong Lei and Shaowei Cai. Solving (weighted) partial maxsat by dynamic local search for SAT. In Jérôme Lang, editor, *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1346–1352. ijcai.org, 2018. doi:10.24963/ijcai.2018/187.
- 22 Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In Maria Fox and David Poole, editors, *Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. doi:10.1609/aaai.v24i1.7536.
- 23 Longlong Lin, Pingpeng Yuan, Rong-Hua Li, and Hai Jin. Mining diversified top- r lasting cohesive subgraphs on temporal networks. *IEEE Transactions on Big Data*, 8(6):1537–1549, 2022. doi:10.1109/TBDATA.2021.3058294.

- 24 Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. Finding top- k shortest paths with diversity. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):488–502, 2018. doi:10.1109/TKDE.2017.2773492.
- 25 Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. Maximum and top- k diversified biclique search at scale. *The VLDB Journal*, 31(6):1365–1389, 2022. doi:10.1007/s00778-021-00681-6.
- 26 Ruben Martins, Norbert Manthey, Miguel Terra-Neves, Vasco Manquinho, and Inês Lynce. Open-WBO@ MaxSAT evaluation 2020. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 24–25, 2020.
- 27 Christian J. Muise, Sheila A. McIlraith, and J. Christopher Beck. Optimally relaxing partial-order plans with MaxSAT. In Lee McCluskey, Brian Charles Williams, José Reinaldo Silva, and Blai Bonet, editors, *Proceedings of the 22nd International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI, 2012. doi:10.1609/icaps.v22i1.13537.
- 28 Alexander Nadel. Generating diverse solutions in SAT. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing – SAT 2011 – 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2011. doi:10.1007/978-3-642-21581-0_23.
- 29 Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. An efficient algorithm for computing the distance between close partitions. *Discrete Applied Mathematics*, 159(1):53–59, 2011. doi:10.1016/j.dam.2010.09.002.
- 30 Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. Spacing memetic algorithms. In Natalio Krasnogor and Pier Luca Lanzi, editors, *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, pages 1061–1068. ACM, 2011. doi:10.1145/2001576.2001720.
- 31 Caleb Voss, Mark Moll, and Lydia E. Kavvaki. A heuristic approach to finding diverse short paths. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 4173–4179. IEEE, 2015. doi:10.1109/ICRA.2015.7139774.
- 32 Jun Wu, Chu-Min Li, Lu Jiang, Junping Zhou, and Minghao Yin. Local search for diversified top- k clique search problem. *Computers & Operations Research*, 116:104867, 2020. doi:10.1016/j.cor.2019.104867.
- 33 Jun Wu, Chu-Min Li, Luzhi Wang, Shuli Hu, Peng Zhao, and Minghao Yin. On solving simplified diversified top- k s -plex problem. *Computers & Operations Research*, 153:106187, 2023. doi:10.1016/j.cor.2023.106187.
- 34 Jun Wu, Chu-Min Li, Yupeng Zhou, Minghao Yin, Xin Xu, and Dangdang Niu. HEAD: A hybrid evolutionary algorithm for diversified top- k weight clique search problem. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 4821–4827. International Joint Conferences on Artificial Intelligence Organization, July 2022. doi:10.24963/ijcai.2022/668.
- 35 Mingyu Xiao. An exact MaxSAT algorithm: Further observations and further improvements. In Luc De Raedt, editor, *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 1887–1893. ijcai.org, 2022. doi:10.24963/ijcai.2022/262.
- 36 Hongfei Xu, Yu Gu, Jianzhong Qi, Jiayuan He, and Ge Yu. Diversifying top- k routes with spatial constraints. *Journal of Computer Science and Technology*, 34(4):818–838, 2019. doi:10.1007/s11390-019-1944-6.
- 37 Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Diversified top- k clique search. *The VLDB Journal*, 25(2):171–196, 2016. doi:10.1007/s00778-015-0408-z.
- 38 Xiaoqi Zheng, Taigang Liu, Zhongnan Yang, and Jun Wang. Large cliques in arabidopsis gene coexpression network and motif discovery. *Journal of plant physiology*, 168(6):611–618, 2011. doi:10.1016/j.jplph.2010.09.010.