

An Exact Algorithm Based on MaxSAT Reasoning for the Maximum Weight Clique Problem

Zhiwen Fang

ZHIWENF@GMAIL.COM

*State Key Lab. of Software Development Environment, Beihang University, China
MIS, Universit de Picardie Jules Verne, Amiens, France*

Chu-Min Li

CHU-MIN.LI@U-PICARDIE.FR

MIS, Universit de Picardie Jules Verne, Amiens, France

Ke Xu

KEXU@NLSDE.BUAA.EDU.CN

State Key Lab. of Software Development Environment, Beihang University, China

Abstract

Recently, MaxSAT formalisms are shown very effective for solving combinatorial optimization problems. An important MaxSAT formalism is weighted partial MaxSAT dealing with hard clauses, which must be satisfied in all solutions, and soft clauses, which are weighted and can be falsified. In this paper, we apply weighted partial MaxSAT to solve the maximum weight clique (MWC) problem. We first study and discuss three encodings from MWC into MaxSAT. The drawbacks of these encodings motivate us to propose a new encoding of MWC into a special weighted partial MaxSAT, called LW (Literal-Weighted) encoding, in which both soft clauses and literals in soft clauses are weighted. In order to make the LW encoding work, we extend MaxSAT reasoning by introducing two notions called *Top-k* literal failed clause and *Top-k* empty clause respectively, as well as two transformation rules to split an LW clause when necessary, to handle both clause weights and literal weights and thus to compute a tight upper bound for such a MaxSAT instance, which is also an upper bound for the encoded MWC instance. We implement an exact branch-and-bound algorithm MWCLQ based on the extended MaxSAT reasoning. To the best of our knowledge, it is the first time that MaxSAT reasoning techniques are applied specifically to compute a tight upper bound in a branch-and-bound algorithm for MWC. Experimental evaluations on the broadly used DIMACS benchmark, BHOSLIB benchmark, random graphs and the benchmark from the winner determination problem show that our approach allows MWCLQ to reduce the search space significantly and to solve MWC instances effectively. Consequently, MWCLQ outperforms state-of-the-art exact algorithms on the vast majority of instances. Moreover, it is surprisingly effective in solving hard and dense instances.

1. Introduction

Consider an undirected graph $G = (V, E)$, where V is a set of n vertices $\{v_1, v_2, \dots, v_n\}$ and E is a set of m edges. The density of G is computed as $2m/(n(n-1))$. A clique of G is a subset $C \subseteq V$ in which every pair of vertices is adjacent. On the contrary, an independent set of G is a subset $I \subseteq V$ in which every pair of vertices is disconnected. A vertex cover of G is a subset $S \subseteq V$ such that every edge in G has at least one endpoint in S . The maximum clique (MC) problem asks to find a clique with the largest cardinality. The MC problem is a prominent combinatorial optimization problem and it is tightly related to two other well-known graph problems, namely the maximum independent set (MIS) problem and the minimum vertex cover (MVC) problem. Concretely, a maximum clique C of G is a maximum independent set of the complement graph \overline{G} of G , and $V \setminus C$ is a minimum vertex cover of \overline{G} . Therefore, algorithms for any of the three problems can directly

be applied to solve the others in practice. In addition, most exact MC solvers take advantage of the following relation between the size of a maximum clique and the number of independent sets. If G can be partitioned into k independent sets, then G cannot contain a clique larger than k , because an independent set can contribute at most one vertex to a clique.

The MC problem is NP-hard and its decision problem is NP-complete (Karp, 1972), which appears in many applications such as social network analysis (Zhang, Nie, Jiang, Chen, & Liu, 2014b; Kibanov, Atzmueller, Scholz, & Stumme, 2014). It is also fixed-parameter intractable (Downey & Fellows, 1995). Moreover, it is proved that approximating MC within $|V|^{1-\varepsilon}$ for any given $\varepsilon > 0$ is NP-hard (Zuckerman, 2006). The best polynomial-time approximation algorithm achieves an approximation ratio of $O(n(\log \log n)^2/(\log n)^3)$ (Feige, 2004). Because of the theoretical and practical importance of the MC problem, a huge amount of effort has been devoted to solve it by designing two types of algorithms (also called solvers). One type is heuristic algorithms mainly including stochastic local search, e.g., (Pullan & Hoos, 2006; Cai, Su, & Sattar, 2011; Cai, Su, Luo, & Sattar, 2013; Fang, Chu, Qiao, Feng, & Xu, 2014a). Another is exact algorithms including branch-and-bound (BnB) search, e.g., (Östergård, 2002; Régim, 2003; Tomita & Seki, 2003; Konc & Janezic, 2007; Li & Quan, 2010b; Tomita & Kameda, 2007; Li, Fang, & Xu, 2013). Heuristic algorithms are able to solve large-scale instances but cannot guarantee the optimality of their solutions. Exact algorithms guarantee the optimality of their solutions, but the worst-case time complexity is exponential unless $P = NP$.

A tight upper bound of the size of a maximum clique in a graph is essential for a BnB algorithm to solve the MC problem efficiently. However, it is very challenging to obtain such an upper bound in reasonable time. Most state-of-the-art BnB algorithms apply approximation coloring and independent set partition algorithms to compute an upper bound for MC. For instance, Fahle (Fahle, 2002) uses the constructive heuristic DSATUR to color vertices one by one according to their degrees. The authors in (Tomita & Seki, 2003; Konc & Janezic, 2007; Tomita & Kameda, 2007; Li & Quan, 2010b) apply the greedy strategy proposed in (Tomita & Seki, 2003) to partition the graph into independent sets, and use the number of independent sets in the partition as an upper bound for MC. MaxCLQ (Li & Quan, 2010b, 2010a) encodes an MC instance into a partial MaxSAT instance and improves the upper bound based on the independent set partition by making use of MaxSAT reasoning. The excellent performance of MaxCLQ shows that MaxSAT reasoning technologies allows to compute a tight upper bound for MC within reasonable time. IncMaxCLQ (Li et al., 2013) combines an incremental upper bound and MaxSAT reasoning to compute a tight upper bound more efficiently. In addition to the independent set partition and MaxSAT reasoning, other approaches, such as the graph matching (Régim, 2003), are also used in the upper bounding for MC.

One generalization of the MC problem is to associate each vertex with a positive weight. The weight of a clique is defined as the total weight of vertices in it. The maximum weight clique (MWC) problem consists in finding a clique with the largest weight. It is computationally equivalent to problems like the weighted set packing problem. The MWC problem appears in a variety of real-world applications, such as protein structure predictions (Mascia, Cilia, Brunato, & Passerini, 2010), coding theory (Zhian, Sabaei, Javan, & Tavallaie, 2013), combinatorial auctions (Wu & Hao, 2015), computer vision (Ma & Latecki, 2012; Zhang, Javed, & Shah, 2014a), etc. For example, in the video object segmentation problem, a challenging task is to select a region having high objectness score and sharing similar appearance, which can be solved by an MWC algorithm (Ma & Latecki, 2012).

Compared to the MC problem, less work has been done to solve the MWC problem. Cliquer (Östergård, 2002, 2001) is one of the few state-of-the-art exact solvers for both MC and MWC, and it deals with MC and MWC using similar methods. In a preprocessing, Cliquer partitions the graph into independent sets by determining one independent set at a time. As long as there are vertices that can be added into the current independent set, the one with the largest degree is added. The purpose of this preprocessing is to define a vertex ordering $v_1 < v_2 < \dots < v_n$, where v_i ($1 \leq i \leq n$) is the vertex inserted into an independent set at time i . Then Cliquer searches for an MC or MWC in the subgraph induced by $\{v_i, v_{i+1}, \dots, v_n\}$ successively for $i=n, n-1, \dots, 1$ (in this ordering). The MC or the MWC in the subgraph induced by $\{v_i, v_{i+1}, \dots, v_n\}$ is associated with v_i and is used to prune subtrees in subsequent search. Kumlander (Kumlander, 2004, 2008) inherits the search strategy from Cliquer. In addition to the MWC associated with v_i , Kumlander also partitions the current subgraph into independent sets and uses the sum of the maximum weight in each independent set as an upper bound. VCTable (Shimizu, Yamaguchi, Saitoh, & Masuda, 2012) improves Kumlander’s algorithm by a new initial vertex order and a better implementation using bitwise operations. Yamaguchi and Masuda (Yamaguchi & Masuda, 2008) propose a new upper bound based on the longest path in a directed acyclic graph constructed from the original graph, which improves the bound based on the independent set partition. OTCliquer (Shimizu, Yamaguchi, Saitoh, & Masuda, 2013), which is also based on Cliquer, uses a dynamic programming strategy to calculate upper bounds of some small subproblems in a preprocessing, and stores all the results in a table. The stored upper bounds are used during search. MinSAT (Li, Zhu, Manyà, & Simon, 2012) is an exact solver for the MinSAT problem. An important application of MinSAT is to solve combinatorial optimization problems such as MC and MWC. MinSAT constructs a weighted graph for the MinSAT instance and uses the clique partition combined with MaxSAT reasoning to compute a tight bound for the MinSAT instance to solve. In addition to exact algorithms, some heuristic algorithms are also proposed to solve the MWC problem (Pullan, 2008; Wu, Hao, & Glover, 2012; Benlic & Hao, 2013).

In this paper, we apply MaxSAT reasoning to solve MWC. We first study three usual encodings of MWC into MaxSAT. All these encodings have intrinsic difficulties in dealing with vertex weights. This motivates us to propose a new encoding from MWC into MaxSAT called the LW (Literal-Weighted) encoding, in which both soft clauses and literals in soft clauses are weighted. We then introduce two notions called the *Top-k* literal failed clause and the *Top-k* empty clause respectively, and two transformation rules to handle both clause weights and literal weights. The key idea of our approach is that the handling of the weight of a literal is driven by conflicts. To the best of our knowledge, it is the first time that MaxSAT reasoning techniques are used specifically in the literature to compute a tight upper bound in a BnB algorithm for MWC, which allows us to develop a highly efficient MWC algorithm called MWCLQ. Experimental results on the widely used DIMACS benchmark, BHOSLIB benchmark, random graphs and realistic benchmark from the winner determination problem show that MWCLQ reduces the search space significantly and outperforms the state-of-the-art exact algorithms on the vast majority of instances from those benchmarks.

This paper is extended from (Fang, Li, Qiao, Feng, & Xu, 2014b) by:

- clearly motivating the LW encoding by illustrating the weakness of three classical encodings of MWC into MaxSAT in dealing with vertex weights of a graph;
- introducing the notion *Top-k* empty clause and exploiting it in the upper bounding for MWC;

- formally proving the transformation rules;
- adding more experimental results to show the effectiveness of our approach. The algorithm MWCLQ is now compared with the integer programming solver CPLEX, and the MinSAT solver MinSatz, especially on realistic instances from the winner determination problem. State-of-the-art MaxSAT solvers using different encodings of MWC into MaxSAT are also compared with MWCLQ. In order to further evaluate the LW encoding, which the state-of-the-art MaxSAT solvers cannot use, we compare different versions of MWCLQ, in which the only difference is the encoding of MWC into MaxSAT used to compute the upper bound.

This paper is organized as follows. In the next section, we introduce some necessary notations and background knowledge. In Section 3, we present MWCLQ and different encodings of MWC into MaxSAT, before extending MaxSAT reasoning to weighted literals by introducing the notions of *Top-k* literal failed clause and *Top-k* empty clause, and two transformation rules. Experimental results are shown in Section 4. Section 5 concludes the paper.

2. Preliminaries

The subgraph of G induced by a subset $V' \subseteq V$ is $G' = (V', E')$, where $E' = \{\{v_i, v_j\} \mid v_i, v_j \in V' \wedge \{v_i, v_j\} \in E\}$. For a vertex v , $\Gamma(v) = \{u \mid \{u, v\} \in E\}$ is the set of neighbors of v and the cardinality $|\Gamma(v)|$ is called the degree of v . We use G_v to denote the subgraph induced by $\Gamma(v) \cup \{v\}$ and $G \setminus v$ to denote the one induced by $V \setminus \{v\}$. A maximal clique is a clique that cannot be extended any more. A maximum clique is a maximal clique of the largest possible size. The cardinality of a maximum clique of G is usually denoted as $\omega(G)$ and is called the *clique number* of G . For any vertex v in G , a maximum clique of G can be either in G_v or in $G \setminus v$. The chromatic number of G , denoted by $\chi(G)$, is the minimum number of colors needed to color the vertices of G such that no two adjacent vertices share the same color. The vertices sharing the same color constitute an independent set. Therefore, the graph coloring problem is equivalent to partitioning V into a minimum number of independent sets. Note that $\chi(G)$ is greater than or equal to $\omega(G)$.

A graph can be edge-weighted or vertex-weighted. We focus on the vertex-weighted graph in this paper. Formally, a vertex-weighted undirected graph $G = (V, E, w)$ is an undirected graph $G = (V, E)$ combined with a weighting function $w: V \rightarrow R^+$ such that every vertex v is associated with a positive weight $w(v)$. In the sequel, we use the term *weighted graph* instead of vertex-weighted graph for simplicity. The weight of a clique C in G is defined to be $w(C) = \sum_{v \in C} w(v)$. Given a weighted graph G , the maximum weight clique problem asks to find a clique with the largest weight in G (Östergård, 2001; Pullan, 2008; Wu et al., 2012) and this largest weight is often denoted by $\omega_v(G)$ in the literature. Note that a maximum weight clique is not necessarily a clique containing the maximum number of vertices, but it must be a maximal clique

The MC problem can be encoded into a partial MaxSAT problem. In MaxSAT, a variable x may take value 0 (*false*) or 1 (*true*). A literal ℓ is a variable x or its negation \bar{x} . A clause $c = \ell_1 \vee \ell_2 \vee \dots \vee \ell_l$ is a disjunction of literals, which can also be expressed as a set $\{\ell_1, \ell_2, \dots, \ell_l\}$. A clause is satisfied if and only if the clause has at least one literal assigned to *true*. The length of clause c is the number of literals it contains, denoted by $length(c)$. A unit clause is a clause containing only one literal. An empty clause, denoted by \square , contains no literals and cannot be satisfied. A conjunctive normal form (CNF) formula $\phi = c_1 \wedge c_2 \wedge \dots \wedge c_m$ is a conjunction of clauses. Given a CNF formula ϕ on the set of variables $\{x_1, x_2, \dots, x_n\}$, the maximum satisfiability (MaxSAT) problem is to find

an assignment satisfying the maximum number of clauses. The minimum satisfiability (MinSAT) problem, on the contrary, is to find an assignment minimizing the number of satisfied clauses (Li et al., 2012). In some cases, some clauses can be declared to be hard and must be satisfied in all solutions, and other clauses are soft and can be falsified. The partial MaxSAT problem asks to find an assignment to maximize the number of satisfied soft clauses while satisfying all hard clauses. Note that the MaxSAT problem is a particular partial MaxSAT problem without hard clauses. A weighted clause is a pair (c, w) , where c is a soft clause and w , a positive number, is its weight. A weighted partial MaxSAT problem is to find an assignment maximizing the total weight of satisfied soft clauses while satisfying all hard clauses. A weighted MaxSAT problem is a weighted partial MaxSAT problem without hard clauses. The optimal solution of a (weighted) (partial) MaxSAT instance ϕ is denoted as $opt(\phi)$ in this paper.

To maximize the number of satisfied clauses equals to minimize the number of falsified clauses. Many algorithms based on the BnB scheme for MaxSAT compute a lower bound of the number of falsified clauses (Li, Manyà, & Planes, 2006; Li, Manyà, & Planes, 2007; Larrosa, Heras, & de Givry, 2008; Kügel, 2010) to prune search space when solving a MaxSAT instance ϕ . Detecting disjoint inconsistent subsets is proved to be very powerful in computing such a bound, where a subset of soft clauses is said *inconsistent* if this subset together with hard clauses is unsatisfiable. Unit propagation is an effective technique widely used in SAT and MaxSAT solvers (Li & Anbulagan, 1997; Li, Manyà, & Planes, 2005). The pseudo-code allowing to find an inconsistent subset of soft clauses based on unit propagation is given in Algorithm 1 (Li et al., 2005). The algorithm works as follows. It uses a stack S to store all unit clauses of ϕ and performs unit propagation until an empty clause is produced or all unit clauses in S are propagated. It is called iteratively to find as many disjoint inconsistent subsets of soft clauses as possible. Note that soft clauses involved in an inconsistent subset are removed before detecting other inconsistent subsets to ensure that these subsets are disjoint.

Algorithm 1: ConflictDetectionByUP(ϕ, S), to detect an inconsistent subset of soft clauses.

Input: MaxSAT instance ϕ and a stack S storing all unit clauses of ϕ .

Output: Return *True* if unit propagation results in an empty clause, otherwise *False*.

```

1 begin
2   while  $S$  is not empty do
3     pop a unit clause  $u$  from  $S$ ;
4      $\ell \leftarrow$  the only literal in  $u$ ;
5     foreach clause  $c \in \phi$  contains  $\ell$  do
6        $\lfloor$  satisfy  $c$ ;
7     foreach clause  $c \in \phi$  contains  $\bar{\ell}$  do
8       remove  $\bar{\ell}$  from  $c$ ;
9       if  $c$  is empty then
10         $\lfloor$  return True;
11       if  $c$  is a unit clause then
12         $\lfloor$  push  $c$  into  $S$ ;
13 return False;

```

Failed literal detection (Freeman, 1995) is used to enhance unit propagation in MaxSAT solving (Li et al., 2006). A literal ℓ of a CNF formula ϕ is called a *failed literal* of ϕ , if unit propagation in $\phi \cup \{\ell\}$ results in an empty clause. Let $IS(\ell)$ be the set of soft clauses used in the unit propagation to derive an empty clause after assigning *true* to ℓ . If all literals in a soft clause $c = \{\ell_1, \ell_2, \dots, \ell_l\}$ are failed, then $\{c\} \cup IS(\ell_1) \cup IS(\ell_2) \cup \dots \cup IS(\ell_l)$ is an inconsistent subset of soft clauses. Observe that for the MaxSAT instance encoding an MC instance, one does not detect whether or not a negative literal is failed, because a variable only has one positive occurrence in a soft clause. Please refer to (Li & Quan, 2010b) for more details.

3. MWCLQ: An Exact Algorithm for MWC

In this section, we propose an exact algorithm based on the BnB scheme, namely MWCLQ, for MWC. We first describe a basic BnB algorithm. Then we introduce an approach to encode an MWC instance into a literal-weighted MaxSAT instance. Finally, we show how to compute a tight upper bound for MWC using MaxSAT reasoning.

3.1 Branch-and-bound Search for MWC

Algorithm 2 depicts the pseudocode of MWCLQ.

Algorithm 2: MWCLQ(G, C, LB), a branch-and-bound algorithm for MWC.

Input: A weighted graph $G=(V, E, w)$, a clique C under construction, and a lower bound LB .

Output: A clique with weight greater than LB , \emptyset if no such clique is found.

```

1 begin
2   if  $|V| = 0$  then
3     return  $C$ ;
4    $UB \leftarrow overestimate(G) + w(C)$ ;
5   if  $UB \leq LB$  then
6     return  $\emptyset$ ;
7    $v \leftarrow select(V)$ ;
8    $C_1 \leftarrow MWCLQ(G_v \setminus v, C \cup \{v\}, LB)$ ;
9    $LB \leftarrow max(LB, w(C_1))$ ;
10   $C_2 \leftarrow MWCLQ(G \setminus v, C, LB)$ ;
11  if  $w(C_1) \leq w(C_2)$  then
12    return  $C_2$ ;
13  else
14    return  $C_1$ ;

```

MWCLQ searches for a clique, extended from C , with weight larger than LB in $G = (V, E, w)$. It first computes an upper bound UB by calling $overestimate(G)$ and then compares UB with LB to test whether further search is necessary in G . If it is possible to find a better solution, MWCLQ selects a vertex by calling $select(V)$. For any vertex v in G , a maximum weight clique of G is

either a clique in G_v containing v or a clique in $G \setminus v$ not containing v . Thus, MWCLQ searches for a maximum weight clique in G_v and in $G \setminus v$ successively.

MWCLQ has a form similar to MaxCLQ, a BnB algorithm for MC (Li & Quan, 2010b), but the *overestimate* function for computing an upper bound, and the *select* function for choosing a branching vertex, are significantly different. These two functions are essential for both MWCLQ and MaxCLQ. A high-quality upper bound allows solvers to prune useless search, and a good vertex ordering promises an efficient search process. MaxCLQ computes a base upper bound by partitioning G into independent sets, and then improves the base upper bound by MaxSAT reasoning. Meanwhile, MaxCLQ orders vertices by selecting first the vertex with minimum degree. In this paper, we use a simple vertex ordering to select v with the largest weight, breaking ties in favor of the vertex with higher degree, and focus on how to efficiently compute a tight upper bound based on MaxSAT reasoning.

3.2 Encoding MWC into MaxSAT

An MC or MWC instance can be encoded into MaxSAT as follows.

Boolean variables: A boolean variable x_i is added for each vertex v_i , which is assigned the value *true* if and only if v_i is in the maximum clique under construction;

Hard clauses: A set of hard clauses is added to require that any pair of unconnected vertices does not belong to the same clique. Concretely, a hard clause $\bar{x}_i \vee \bar{x}_j$ is added for each pair of unconnected vertices v_i and v_j ;

Soft clauses: There exist different ways to define the set of soft clauses. Given a graph, all encodings of MC or MWC into MaxSAT presented in this paper use the same set of hard clauses. They differ only in soft clauses, and will be analyzed and discussed in this subsection.

An MC instance can also be encoded into a MinSAT instance without hard clauses by adopting the approach proposed in (Ignatiev, Morgado, & Marques-Silva, 2014). The obtained MinSAT instance can be in turn encoded into a MaxSAT instance without hard clauses using the approaches presented in (Kügel, 2012; Zhu, Li, Manyà, & Argelich, 2012). This encoding without hard clauses provides a new angle of view for the MC or MWC solving, which awaits future research. In this paper, we focus on encodings of MWC into MaxSAT with hard clauses and only different in soft clauses.

3.2.1 DIRECT ENCODING OF MWC INTO MAXSAT

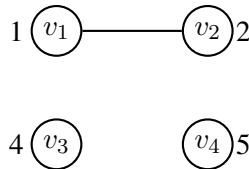


Figure 1: A weighted graph with 4 vertices and 1 edge. Numbers indicate vertex weights.

A straightforward way to define soft clauses is to associate a unit soft clause x_i with each vertex v_i , giving the *direct encoding* of MC or MWC into MaxSAT. For example, the MC instance for the graph in Fig. 1 (without considering vertex weights) is encoded into the following partial MaxSAT instance: (1) the set of variables is $\{x_1, x_2, x_3, x_4\}$; (2) the set of hard clauses is $\{\bar{x}_1 \vee \bar{x}_3, \bar{x}_1 \vee \bar{x}_4, \bar{x}_2 \vee \bar{x}_3, \bar{x}_2 \vee \bar{x}_4, \bar{x}_3 \vee \bar{x}_4\}$; and (3) the set of soft clauses is $\{x_1, x_2, x_3, x_4\}$.

Any assignment satisfying all hard clauses gives rise to a clique, since the variables assigned the value *true* correspond to pairwise connected vertices. For the non-weighted case, an assignment satisfying all hard clauses and maximizing the number of satisfied soft clauses gives rise to a maximum clique.

For the weighted case, each unit soft clause is associated with the weight of the corresponding vertex. For example, the MWC instance for the weighted graph in Fig. 1 can be encoded into a weighted partial MaxSAT instance with the same boolean variables and the same hard clauses as in the MC instance. The set of weighted soft clauses is: $\{(x_1,1), (x_2,2), (x_3,4), (x_4,5)\}$. An assignment satisfying all hard clauses and maximizing the total weight of satisfied soft clauses gives rise to a maximum weight clique.

Once encoded into MaxSAT, MaxSAT reasoning can be applied to solve the MC or MWC problem. Many MaxSAT algorithms obtain an upper bound of satisfied soft clauses by computing a lower bound of the number of falsified soft clauses. Inconsistent subsets of soft clauses are often used for computing such a lower bound. Recall that a subset of soft clauses is said *inconsistent* if the subset together with hard clauses is not satisfiable. For a non-weighted MaxSAT instance ϕ with m soft clauses, if r disjoint inconsistent subsets of soft clauses are detected, then $m-r$ is an upper bound of $opt(\phi)$ (Li et al., 2005, 2006).

For the weighted case, we define the weight of a subset S of soft clauses to be the minimum weight of clauses in S , namely,

$$w(S) = \min_{c \in S} w(c).$$

Similar to the non-weighted case, we have the following proposition.

Proposition 1. (Li et al., 2007) *Given a weighted partial MaxSAT instance ϕ , if s disjoint inconsistent subsets S_1, S_1, \dots, S_s are detected in ϕ , then $opt(\phi) \leq \sum_{\text{soft clause } c \in \phi} w(c) - \sum_{1 \leq i \leq s} w(S_i)$.*

Observe that unit soft clauses of direct encoding do not capture any connection between vertices, because the same set of soft clauses is used for every graph with n vertices, no matter how these vertices are connected between each other. For an MC instance, the number of soft clauses is used as a base upper bound, and then MaxSAT reasoning is applied to improve the base upper bound by detecting inconsistent soft clause subsets. Unfortunately, direct encoding can only give a trivial base upper bound, which is the number of vertices in the graph. Moreover, since all soft clauses of direct encoding are unit, an inconsistent subset contains exactly two soft clauses, limiting the improvement of the upper bound to the half of the number of vertices in the graph. The weighted case is similar. The base upper bound is the total weight of all vertices. We use the following example to illustrate that the direct encoding cannot compute a tight upper bound even for a very simple MWC instance.

Example 1. *The total weight of soft clauses of direct encoding for the graph in Fig. 1 is 12. Using unit propagation, $\{(x_3, 4), (x_4, 5)\}$ is found to be an inconsistent subset because of the hard clause $\bar{x}_3 \vee \bar{x}_4$, then the upper bound can be improved by 4. The left soft clauses are: $(x_1, 1), (x_2, 2),$*

$(x_4, 1)$, then unit propagation detects that $\{(x_1, 1), (x_4, 1)\}$ is also inconsistent, then we can improve the upper bound by 1. Finally, the upper bound is improved to 6, larger than the optimal solution 5.

Because of these drawbacks, the direct encoding does not perform well, as shown by the experimental results presented in Section 4.3. One might want to add an at-most-one constraint for each independent set to remedy the drawbacks of direct encoding. That is, for each independent set $I = \{v_1, v_2, \dots, v_l\}$, add the at-most-one constraint $x_1 + x_2 + \dots + x_l \leq 1$. However, the at-most-one constraint does not add anything new, because the subset of hard clauses $\{\bar{x}_i \vee \bar{x}_j \mid 1 \leq i < j \leq l\}$ in the encoding already enforces the at-most-one constraint (Chen, 2010). In addition, the encoding of the at-most-one constraint using hard binary clauses is efficient enough when the independent set is not extremely large, which is usually the case when solving hard MC or MWC instances. Therefore, the at-most-one constraint is presumably useless in the encoding from MC or MWC into MaxSAT.

3.2.2 SPLIT ENCODING AND ITERATIVE SPLIT ENCODING OF MWC INTO MAXSAT

Another encoding is introduced for MC in MaxCLQ (Li & Quan, 2010b) by defining the set of soft clauses based on an independent set partition of G . Concretely, MaxCLQ first partitions G into a set of independent sets, then creates a soft clause for each independent set, which is a disjunction of the variables corresponding to the vertices in the independent set. The independent set based encoding is shown substantially more efficient than the direct encoding to solve MC.

A natural way to extend the independent set based MaxSAT encoding to MWC is to split vertex weights. For an independent set $I = \{v_1, v_2, \dots, v_l\}$, where $w(v_1) \geq w(v_2) \geq \dots \geq w(v_l)$, we can split vertex weights in I by the minimum weight $w(v_l)$, thus we obtain a weighted soft clause $(x_1 \vee x_2 \vee \dots \vee x_l, w(v_l))$ and l' unit soft clauses: $(x_1, w(v_1) - w(v_l))$, $(x_2, w(v_2) - w(v_l))$, \dots , $(x_{l'}, w(v_{l'}) - w(v_l))$, where l' is the largest integer such that $w(v_{l'}) > w(v_l)$ in I . This encoding is called the *split encoding*. The total weight of soft clauses in the split encoding should be less than that of the direct encoding, giving a better base upper bound, but it may still be large.

Example 2. A possible independent set partition of the graph in Fig. 1 is $\{v_4, v_3, v_1\}$ and $\{v_2\}$. The soft clauses of split encoding based on the partition are $(x_4 \vee x_3 \vee x_1, 1)$, $(x_4, 4)$, $(x_3, 3)$ and $(x_2, 2)$. The total weight of the soft clauses is 10, which is better than that of the direct encoding. Using unit propagation, $\{(x_4, 4), (x_3, 3)\}$ is found to be an inconsistent subset because of $\bar{x}_3 \vee \bar{x}_4$, which allows us to improve the upper bound by 3. In the same way, $\{(x_4, 1), (x_2, 2)\}$ is found to be inconsistent, which improves the upper bound by 1. Finally, the remaining soft clauses $(x_4 \vee x_3 \vee x_1, 1)$ and $(x_2, 1)$ are consistent. Thus, the upper bound computed using the split encoding is still 6.

The split encoding can be improved by the so-called the *iterative split encoding*, as used in MinSatz (Li et al., 2012). The idea of the iterative split encoding is to split vertex weights repeatedly until all vertices in the same independent set have the same weight. Concretely, for an independent set $I = \{v_1, v_2, \dots, v_l\}$ such that $w(v_1) \geq w(v_2) \geq \dots \geq w(v_l)$, we add a soft clause $(x_1 \vee x_2 \vee \dots \vee x_l, w(x_l))$, and repeatedly find the largest l' such that $w(x_{l'}) > w(x_l)$ and add a soft clause $(x_1 \vee x_2 \vee \dots \vee x_{l'}, w(x_{l'}) - w(x_l))$, while $l' \geq 1$.

Example 3. A possible independent set partition of the graph in Fig. 1 is $\{v_4, v_3, v_1\}$, $\{v_2\}$. The soft clauses of the iterative split encoding based on the partition are $(x_4 \vee x_3 \vee x_1, 1)$, $(x_4 \vee x_3, 3)$, $(x_4, 1)$ and $(x_2, 2)$. The total weight of soft clauses is 7. Starting unit propagation by setting $x_4 = 1$,

we find that $\{(x_4, 1), (x_2, 2)\}$ is an inconsistent subset, so that we can improve the upper bound by 1 and get a new soft clause $(x_2, 1)$. Then, we find that $\{(x_4 \vee x_3, 3), (x_2, 1)\}$ is inconsistent, so the upper bound is improved to 5, which is the tightest upper bound.

The iterative split encoding gives a non-trivial base upper bound, which is better than those obtained by the direct encoding and the split encoding. The key point of the iterative split encoding is that the vertex weights are split in advance, generating numerous soft clauses, such that a variable may appear in several soft clauses and that some clauses may differ only in one variable (e.g. clauses $(x_4 \vee x_3 \vee x_1, 1)$ and $(x_4 \vee x_3, 3)$ in Example 3). Many splittings may not be useful in the upper bound computation, and MaxSAT reasoning may be more complicated in such an instance with numerous soft clauses.

3.2.3 LITERAL-WEIGHTED ENCODING OF MWC INTO MAXSAT

Recall that the MaxSAT encoding for MC in MaxCLQ guarantees that a variable appears only once in soft clauses and the number of soft clauses equals to the number of independent sets. MaxSAT reasoning in such an instance should be much simpler. In order to extend this advantage to MWC and not to split vertex weights in advance, we introduce the *Literal-Weighted encoding* (LW encoding) from MWC into LW MaxSAT, in which the literals in a soft clause are also weighted. Concretely, a weighted literal is a pair (ℓ, w) , where ℓ is a literal and w is its weight. A *literal-weighted soft clause* (LW soft clause) is a disjunction of weighted literals. Formally, given an MWC instance $G = (V, E, w)$, we encode G into a literal-weighted MaxSAT (LW MaxSAT) instance as follows:

- Associate each vertex v_i with a weighted literal $(x_i, w(x_i))$, where $w(x_i) = w(v_i)$;
- Add a hard clause $\bar{x}_i \vee \bar{x}_j$ for each pair of unconnected vertices v_i and v_j ;
- For an independent set $I = \{v_1, v_2, \dots, v_l\}$, add an LW soft clause $c = (x_1, w(v_1)) \vee (x_2, w(v_2)) \vee \dots \vee (x_l, w(v_l))$.

The LW soft clause c can also be presented as a set $\{(x_1, w(x_1)), (x_2, w(x_2)), \dots, (x_l, w(x_l))\}$. The weight of c is defined to be

$$w(c) = \max_{1 \leq j \leq l} w(x_j).$$

An LW clause c is *ordered* if $w(x_1) \geq w(x_2) \geq \dots \geq w(x_l)$. In the sequel, LW soft clauses are always ordered. Therefore, the weight of an LW soft clause is always $w(x_1)$. A classical optimal solution of the LW MaxSAT instance is an assignment satisfying all hard clauses and maximizing the total weight of satisfied soft clauses. Example 4 illustrates how to use the LW encoding to improve the upper bound for MWC.

Example 4. A possible independent set partition of the graph in Fig. 1 is $\{v_4, v_3, v_1\}$ and $\{v_2\}$. The LW soft clauses are $\{(x_4, 5), (x_3, 4), (x_1, 1)\}$ and $\{(x_2, 2)\}$. The total weight of soft clauses is 7, giving the base upper bound as in the iterative split encoding. Unit propagation by setting $x_2 = 1$ makes x_4 and x_3 false through the hard clauses. Then we find that we can split the clause $\{(x_4, 5), (x_3, 4), (x_1, 1)\}$ into $\{(x_4, 2), (x_3, 2)\}$ and $\{(x_4, 3), (x_3, 2), (x_1, 1)\}$, and keep the base upper bound to be 7, but that the splitting allows to derive an inconsistent subset of soft clauses $\{(x_4, 2), (x_3, 2)\}, \{(x_2, 2)\}$ and to improve the base upper bound to 5, the tightest possible upper bound.

Note that unit propagation does not need to derive any empty clause to improve the upper bound in this example. Furthermore, in order to obtain the tightest upper bound, one unit propagation

suffices, while two inconsistent subsets need to be derived in the iterative split encoding in Example 3.

Example 4 suggests that the LW encoding can perform better than other encodings in improving the upper bound for MWC, but that MaxSAT reasoning needs to be extended to exploit the LW encoding for MWC. For this purpose, we will study the relationship between MaxSAT and LW MaxSAT and propose two transformation rules to derive inconsistent subsets of soft clauses in the next subsections.

3.3 Relationship between MaxSAT and LW MaxSAT

Using the LW encoding, we encode an MWC instance G into an LW MaxSAT instance ϕ , in which not only soft clauses are weighted, literals in soft clauses are also weighted. Different from the directing encoding, the split encoding and the iterative split encoding, a classical optimal solution of ϕ may not correspond to a maximum weight clique of the graph (see Example 5). The fact motivates us to define the *literal based optimal solution* for ϕ , which is an assignment satisfying all hard clauses and maximizing the total weight of satisfied literals in ϕ . The maximum total weight of satisfied literals is denoted by $opt_l(\phi)$, while the classical optimal solution of ϕ is denoted by $opt(\phi)$. Observe that at most one literal can be satisfied in any literal-weighted soft clause of ϕ and the vertices corresponding to the satisfied literals are connected for any assignment satisfying all hard clauses. Hence, a literal based optimal solution of ϕ gives a maximum weight clique of G . To distinguish from the literal based optimal solution, the classical optimal solution is also called the *clause based optimal solution*.

Example 5. A possible independent set partition of the graph in Fig. 2 is $\{\{v_1, v_3, v_6\}, \{v_2, v_4\}$ and $\{v_5\}\}$. The set of soft clauses of the LW MaxSAT instance ϕ based on this partition is $\{(x_6, 6), (x_3, 2), (x_1, 1)\}, \{(x_2, 7), (x_4, 3)\}, \{(x_5, 4)\}$. The only literal based optimal solution of ϕ is $\{x_5=1, x_6=1\}$ and $opt_l(\phi)=10$, which gives the maximum weight clique. On the other hand, $\{x_1=1, x_2=1\}$ is a clause based optimal solution of ϕ and $opt(\phi)=13$. Moreover, any clause based optimal solution of ϕ does not correspond to the maximum weight clique in the graph.

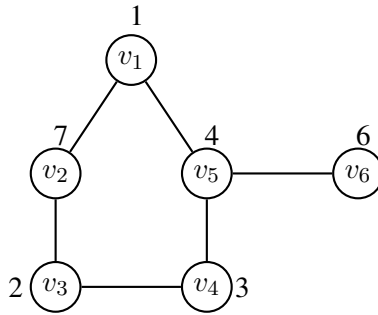


Figure 2: A weighted graph with 6 vertices and 6 edges. Numbers indicate vertex weights.

The following property states a relationship between the literal based optimal solution and the classical optimal solution of an LW MaxSAT instance.

Proposition 2. Consider a weighted graph $G = (V, E, w)$, let ϕ be the LW instance based on an independent set partition of G , then $opt_l(\phi) \leq opt(\phi)$.

Proof. Suppose that $\{x_{i_1}, x_{i_2}, \dots, x_{i_p}\}$ is the set of literals satisfied by a literal based optimal solution of ϕ . Let c_{i_j} be the LW clause containing x_{i_j} . We have

$$opt_l(\phi) = \sum_{j=1}^p w(x_{i_j}) \leq \sum_{j=1}^p w(c_{i_j}) \leq opt(\phi).$$

□

Since $opt_l(\phi)$ is equal to the weight $\omega_v(G)$ of a maximum weight clique, $opt(\phi)$ is an upper bound of $\omega_v(G)$. A trivial upper bound of $opt(\phi)$ is to sum up weights of all soft clauses of ϕ . Hence, a base upper bound of $\omega_v(G)$ in the graph in Fig. 2 is 17 in Example 5. We can easily verify that the maximum weight clique of the graph in Fig. 2 is $\{v_5, v_6\}$ with weight 10. We will define two transformation rules to extend MaxSAT reasoning into LW MaxSAT to improve this base upper bound.

3.4 Transformation Rules for Literal-Weighted MaxSAT

To solve a MaxSAT problem efficiently, it is often useful to transform a MaxSAT instance into an equivalent and simpler instance by applying inference rules. Many inference rules are proposed in the literature (Li et al., 2007; Larrosa et al., 2008; Ansótegui, Bonet, & Levy, 2009). We propose two transformation rules to split an LW soft clause of the MaxSAT instance encoding an MWC instance. The correctness of these rules is based on the following property.

Proposition 3. Let ϕ be an LW MaxSAT instance encoding an MWC instance G . For any LW soft clause $c = \{(x_1, w(x_1)), (x_2, w(x_2)), \dots, (x_l, w(x_l))\}$ of ϕ and $0 < \delta \leq w(x_1)$, split c into $c' = \{(x_1, \delta), (x_2, \min(w(x_2), \delta)), \dots, (x_l, \min(w(x_l), \delta))\}$ and $c'' = \{(x_1, w(x_1) - \delta), (x_2, \max(w(x_2) - \delta, 0)), \dots, (x_l, \max(w(x_l) - \delta, 0))\}$, where literals with weight 0 are removed, we get $\phi' = (\phi \setminus \{c\}) \cup \{c', c''\}$, then $\omega_v(G) \leq opt(\phi') \leq opt(\phi)$.

Proof. First of all, note that both c' and c'' are ordered as c . To prove $\omega_v(G) \leq opt(\phi')$, we first show that $\min(w(x_j), \delta) + \max(w(x_j) - \delta, 0) = w(x_j)$ for any j such that $1 < j \leq l$. In fact,

1. If $w(x_j) \geq \delta$, $\min(w(x_j), \delta) + \max(w(x_j) - \delta, 0) = \delta + w(x_j) - \delta = w(x_j)$;
2. if $w(x_j) < \delta$, $\min(w(x_j), \delta) + \max(w(x_j) - \delta, 0) = w(x_j) + 0 = w(x_j)$.

In other words, the total weight of a literal is not changed by splitting c into c' and c'' . Let $C = \{v_{i_1}, v_{i_2}, \dots, v_{i_p}\}$ be a maximum weight clique of G . Then $\{x_{i_1} = 1, x_{i_2} = 1, \dots, x_{i_p} = 1\}$ with other variables being 0, is an assignment of ϕ' satisfying all hard clauses. Let $S(x_{i_j}) = \{c \mid x_{i_j} \in c\}$ be the set of soft clauses containing x_{i_j} of ϕ' for $1 \leq j \leq p$. All clauses in $S(x_{i_j})$ are satisfied by x_{i_j} . At the same time,

$$\sum_{c \in S(x_{i_j})} w(c) = \sum_{c \in S(x_{i_j})} \max_{x \in c} w(x) \geq \sum_{c \in S(x_{i_j})} w(x_{i_j}) = w(x_{i_j}).$$

Hence, we have,

$$\text{opt}(\phi') \geq \sum_{j=1}^p \sum_{c \in \mathcal{S}(x_{i_j})} w(c) \geq \sum_{j=1}^p w(x_{i_j}) = \sum_{j=1}^p w(v_{i_j}) = \omega_v(G).$$

To prove $\text{opt}(\phi') \leq \text{opt}(\phi)$, let \mathcal{A} be any assignment and $w(\phi, \mathcal{A})$ ($w(\phi', \mathcal{A})$) be the total weight of satisfied soft clauses of ϕ (ϕ').

1. If c is not satisfied in ϕ by \mathcal{A} , then c' and c'' also are not satisfied in ϕ' by \mathcal{A} ;
2. If c is satisfied by \mathcal{A} , c' is also satisfied since c' has the same literals as c , but c'' may not be satisfied because the satisfied literal of c may have weight 0 in c'' and may thus be removed from c'' .

We note that $w(c) = w(c') + w(c'')$, so we have $w(\phi, \mathcal{A}) \geq w(\phi', \mathcal{A})$ for any assignment \mathcal{A} , implying $\text{opt}(\phi') \leq \text{opt}(\phi)$. \square

Given a weighted graph $G = (V, E, w)$, let ϕ be an LW MaxSAT instance based on an independent set partition of G , we propose the following two rules to transform ϕ .

1. **δ -Rule** Given an LW soft clause $c = \{(x_1, w(x_1)), (x_2, w(x_2)), \dots, (x_l, w(x_l))\}$ of ϕ and a weight δ such that $0 < \delta \leq w(x_1)$, split c into $c' = \{(x_1, \delta), (x_2, \min(w(x_2), \delta)), \dots, (x_l, \min(w(x_l), \delta))\}$ and $c'' = \{(x_1, w(x_1) - \delta), (x_2, \max(w(x_2) - \delta, 0)), \dots, (x_l, \max(w(x_l) - \delta, 0))\}$, i.e., $\phi' = (\phi \setminus \{c\}) \cup \{c', c''\}$.
2. **(k, δ) -Rule** Given an LW clause $c = \{(x_1, w(x_1)), (x_2, w(x_2)), \dots, (x_l, w(x_l))\}$ of ϕ , an integer $1 \leq k < l$ and a weight δ such that $0 < \delta \leq w(x_1) - w(x_{k+1})$, split c into $c' = \{(x_1, \delta), (x_2, \max(w(x_2) + \delta - w(x_1), 0)), \dots, (x_k, \max(w(x_k) + \delta - w(x_1), 0))\}$, and $c'' = \{(x_1, w(x_1) - \delta), (x_2, \min(w(x_2), w(x_1) - \delta)), \dots, (x_k, \min(w(x_k), w(x_1) - \delta)), (x_{k+1}, w(x_{k+1})), \dots, (x_l, w(x_l))\}$, i.e., $\phi' = (\phi \setminus \{c\}) \cup \{c', c''\}$.

The purpose of the δ -Rule and the (k, δ) -Rule is to split c into a clause c' with weight δ and a clause c'' with weight $w(x_1) - \delta$ without changing the total weight of a literal. The constraint $\delta \leq w(x_1) - w(x_{k+1})$ is to ensure that c'' remains ordered (i.e., $w(x_1) - \delta \geq \min(w(x_2), w(x_1) - \delta) \geq \dots \geq w(x_{k+1}) \geq \dots \geq w(x_l)$). We use an example to illustrate the (k, δ) -Rule. Let c be an LW soft clause $c = \{(x_1, 5), (x_2, 3), (x_3, 2)\}$, (1) If $k=1$ and $\delta=2$, then $c' = \{(x_1, 2)\}$ and $c'' = \{(x_1, 3), (x_2, 3), (x_3, 2)\}$; (2) If $k=2$ and $\delta=3$, then $c' = \{(x_1, 3), (x_2, 1)\}$ and $c'' = \{(x_1, 2), (x_2, 2), (x_3, 2)\}$; (3) If $k=2$ and $\delta=2$, then $c' = \{(x_1, 2), (x_2, 0)\} = \{(x_1, 2)\}$ and $c'' = \{(x_1, 3), (x_2, 3), (x_3, 2)\}$.

The soundness of the δ -Rule and the (k, δ) -Rule can be easily proved using Proposition 3. These two rules can be applied in many possible ways to generate many possible clauses. For example, as a special case, when $\delta = w(x_l)$ in the δ -Rule, $c' = \{(x_1, \delta), (x_2, \delta), \dots, (x_l, \delta)\}$ and at least x_l is removed from c'' since its weight is 0 in c'' . In other words, we can transform ϕ into a classical weighted MaxSAT instance by repeatedly applying the δ -Rule with $\delta = w(x_l)$ to each clause c containing literals with different weights. In this way, we can obtain a classical weighted partial MaxSAT instance φ , in which all literals in any soft clause have the same weight and whose optimal solution gives a maximum weight clique. Moreover, let ϕ_i be the MaxSAT instance obtained after

i applications of the δ -Rule, we have $opt(\phi) \geq opt(\phi_1) \geq opt(\phi_2) \geq \dots \geq opt(\varphi) = \omega_v(G) = opt_l(\phi)$. In fact, φ is an iterative split encoding of G .

Observe that each application of the δ -Rule or the (k, δ) -Rule gives a MaxSAT instance ϕ_i whose optimal solution gives a possibly tighter upper bound of $\omega_v(G)$. Since unrestricted applications of the two rules may not be effective, we restrict their applications to the cases where an inconsistent subset with weight δ can be derived. Concretely, the (k, δ) -Rule is always applied to a clause c in which the k most weighted literals are failed or falsified, allowing to obtain c' from which an inconsistent subset S of soft clauses can be derived, and the δ -Rule is always applied to a clause c in an inconsistent subset of soft clauses. Each application allows to improve an upper bound of $\omega_v(G)$ by a positive weight δ . Note that δ in both rules is strictly greater than 0, because when $\delta = 0$, both rules only generate new empty clauses with weight 0, which cannot improve the upper bound. Details of our approach are given afterwards.

3.5 Upper Bound Based On MaxSAT Reasoning

In this subsection, we first recall how to detect inconsistent subsets using classical MaxSAT reasoning, i.e., unit propagation and failed literal detection, to improve the upper bound for MWC. Then we introduce two notations to extend the classical MaxSAT reasoning. Finally, we present the overestimating algorithm based on the extended MaxSAT reasoning in MWCLQ.

3.5.1 CLASSICAL MAXSAT REASONING IN LW MAXSAT

First of all, we detect inconsistent subsets using unit propagation presented in Algorithm 1.

Example 6. A possible independent set partition of the graph in Fig. 2 is $\{\{v_1, v_4, v_6\}, \{v_2, v_5\}, \{v_3\}\}$, LW soft clauses based on this partition are $c_1 = \{(x_6, 6), (x_4, 3), (x_1, 1)\}$, $c_2 = \{(x_2, 7), (x_5, 4)\}$ and $c_3 = \{(x_3, 2)\}$. When we set $x_3=1$ to satisfy the unit clause c_3 , \bar{x}_3 will be removed from hard clauses $\bar{x}_1 \vee \bar{x}_3$, $\bar{x}_3 \vee \bar{x}_5$, $\bar{x}_3 \vee \bar{x}_6$. Unit clauses \bar{x}_1 , \bar{x}_5 and \bar{x}_6 imply that $x_1=0$, $x_5=0$ and $x_6=0$, respectively. So, both c_1 and c_2 become unit clauses. Accordingly, we have to set $x_4=1$ and $x_2=1$ to satisfy them, falsifying the hard clause $\bar{x}_2 \vee \bar{x}_4$.

Consequently, an inconsistent soft clause subset $S = \{c_1, c_2, c_3\}$ is detected and the weight of the subset is 2. Proposition 1 allows to decrease the upper bound by 2, thus the improved upper bound is 13. Since all soft clauses are involved in the inconsistent subset, we cannot improve the upper bound any more unless we apply the δ -Rule or the (k, δ) -Rule to split the LW soft clauses. Indeed, we can split the soft clauses in S using the δ -Rule based on the following proposition.

Proposition 4. Let $S = \{c_1, c_2, \dots, c_i\}$ be an inconsistent subset of LW soft clauses, if the δ -Rule is applied to split every $c_j \in S$ into c'_j and c''_j with $\delta=w(S)$, then $S' = \{c'_1, c'_2, \dots, c'_i\}$ is still an inconsistent soft clause subset with weight δ .

Recall that each soft clause $c'_j \in S'$ has the same literals as $c_j \in S$ and has weight δ . Therefore S' is an inconsistent soft clause set with weight δ . The purpose of the splitting is to obtain $S'' = \{c''_1, c''_2, \dots, c''_i\}$ to further improve the upper bound.

Example 7. As presented in Example 6, $\{c_1, c_2, c_3\}$ is an inconsistent subset with weight 2. Applying δ -Rule with $\delta=2$, we have $c'_1 = \{(x_6, 2), (x_4, 2), (x_1, 1)\}$, $c''_1 = \{(x_6, 4), (x_4, 1)\}$, $c'_2 = \{(x_2, 2), (x_5, 2)\}$, $c''_2 = \{(x_2, 5), (x_5, 2)\}$, $c'_3 = \{(x_3, 2)\}$ and $c''_3 = \{(x_3, 0)\}$ (c''_3 is removed).

It is easy to see that $\{c'_1, c'_2, c'_3\}$ is an inconsistent subset of soft clauses with weight 2, and that the remaining clauses $\{(x_6, 4), (x_4, 1)\}, \{(x_2, 5), (x_5, 2)\}$ can be used to further improve the upper bound. However, unit propagation cannot be used to improve the upper bound any more because no unit clause exists. Moreover, failed literal detection does not work either because every soft clause contains at least one literal that is not failed. We propose to apply the (k, δ) -Rule to split the *Top-k literal failed clause* defined afterwards to improve the upper bound.

3.5.2 TOP-K FAILED LITERAL DETECTION

Definition 1. An LW soft clause $c = \{(x_1, w(x_1)), (x_2, w(x_2)), \dots, (x_l, w(x_l))\}$ is *Top-k literal failed* if x_1, x_2, \dots, x_k are all failed literals, where $1 \leq k < l$.

We define the *Top-k weight* of an LW soft clause c as $w_k(c) = w(x_1) - w(x_{k+1})$, where $1 \leq k < \text{length}(c)$.

Proposition 5. Consider an LW soft clause $c = \{(x_1, w(x_1)), (x_2, w(x_2)), \dots, (x_l, w(x_l))\}$. If c is *Top-k literal failed* with $k < l$, and $IS(x_j)$ ($1 \leq j \leq k$) is the set of soft clauses making x_j failed, split c into c' and c'' using the (δ, k) -Rule with $\delta = \min(w_k(c), w(IS(x_1)), w(IS(x_2)), \dots, w(IS(x_k)))$, then $\{c'\} \cup IS(x_1) \cup IS(x_2) \cup \dots \cup IS(x_k)$ is an inconsistent subset of soft clauses with weight δ .

As soon as we determine that c is *Top-k literal failed*, we apply the (δ, k) -Rule to split c and use the δ -Rule to split clauses in $IS(x_1) \cup IS(x_2) \cup \dots \cup IS(x_k)$. An inconsistent subset with weight δ is obtained in this way. Observe that when $k=l$, $\{c\} \cup IS(x_1) \cup IS(x_2) \cup \dots \cup IS(x_l)$ is a classical inconsistent subset of soft clauses with weight $\min(w(c), w(IS(x_1)), w(IS(x_2)), \dots, w(IS(x_l)))$ in which each clause is split only using the δ -Rule.

Example 8. Consider the soft clauses produced in Example 7, $c_1 = \{(x_6, 4), (x_4, 1)\}$ and $c_2 = \{(x_2, 5), (x_5, 2)\}$. We test x_2 in c_2 by setting $x_2=1$. To satisfy hard clauses $\bar{x}_2 \vee \bar{x}_6$ and $\bar{x}_2 \vee \bar{x}_4$, we need to set $x_6=0$ and $x_4=0$. Then, $c_1 = \{(x_6, 4), (x_4, 1)\}$ becomes falsified, making x_2 failed. However, x_5 is not failed. Therefore c_2 is *Top-k literal failed* with $k=1$. Applying the (k, δ) -Rule with $k=1$ and $\delta=3$ to c_2 , we have $c'_2 = \{(x_2, 3)\}$ and $c''_2 = \{(x_2, 2), (x_5, 2)\}$. Applying the δ -Rule with $\delta=3$ to c_1 , we get $c'_1 = \{(x_6, 3), (x_4, 1)\}$ and $c''_1 = \{(x_6, 1)\}$. Consequently, we get an inconsistent subset $\{c'_1, c'_2\}$ with weight 3.

As a result, the detection of a *Top-k literal failed clause* allows to improve the upper bound by 3, giving the tightest upper bound 10.

3.5.3 TOP-K EMPTY CLAUSE DETECTION

It is noteworthy that a literal can be declared to be *failed*, only if unit propagation of the literal falsifies all literals of a clause. Sometimes, the propagation of a literal ℓ just makes the most weighted literals in a soft clause c falsified, but not all literals in c . In this case, ℓ cannot be declared to be *failed*, so we cannot improve the upper bound using approaches presented above. However, we can split c using the (k, δ) -Rule to obtain a falsified clause, so that ℓ can be declared to be failed.

Example 9. Consider the weighted graph G in Fig. 3, a possible independent set partition of G is $\{v_6, v_3\}, \{v_2, v_5\}, \{v_4, v_1\}$ and the LW soft clauses are $c_1 = \{(x_6, 6), (x_3, 2)\}$, $c_2 = \{(x_2, 7), (x_5, 4)\}$ and $c_3 = \{(x_4, 3), (x_1, 1)\}$. The total weight of soft clauses of the LW MaxSAT instance based on this partition is 16. On the one hand, the literals with the largest weight in each soft

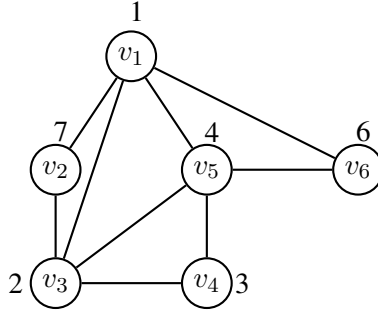


Figure 3: A weighted graph with 6 vertices and 9 edges. Numbers indicate vertex weight

clause are not failed. When we set $x_6=1$, we need to set $x_2=0$ to satisfy the hard clause $\bar{x}_2 \vee \bar{x}_6$. The literal x_6 is not failed, because unit propagation does not falsify any clause. However, the most weighted literal x_2 in c_2 is falsified. We can split c_2 into $c'_2 = \{(x_2, 3)\}$ and $c''_2 = \{(x_2, 4), (x_5, 4)\}$. Then c'_2 is falsified by propagating x_6 . So, x_6 is a failed literal after splitting c_2 , and c_1 is a Top- k ($k=1$) literal failed clause that can be split into $c'_1 = \{(x_6, 3)\}$ and $c''_1 = \{(x_6, 3), (x_3, 2)\}$ using the (k, δ) -Rule. We thus obtain an inconsistent subset $\{c'_1, c'_2\}$ with weight $\delta=3$. Consequently, the upper bound is improved by 3.

Example 9 suggests us to define the Top- k empty clause notion.

Definition 2. An LW soft clause $c = \{(x_1, w(x_1)), (x_2, w(x_2)), \dots, (x_l, w(x_l))\}$ is Top- k empty, where $1 \leq k < l$, if there is a literal ℓ such that when ℓ is assigned to true, unit propagation falsifies the literals x_1, x_2, \dots, x_k in c .

It is straightforward to show the following proposition.

Proposition 6. An inconsistent subset of soft clauses can be derived after splitting a Top- k empty clause by using the (k, δ) -Rule.

3.5.4 OVERESTIMATING ALGORITHM IN MWCLQ

An MWC instance is encoded into an LW MaxSAT instance based on an independent set partition of the weighted graph. However, this partition often is not optimal to derive a tight upper bound for a maximum weight clique of the graph. The δ -Rule, (k, δ) -Rule, Top- k literal failed clause, and Top- k empty clause defined above, allow to extend the classical MaxSAT reasoning, and to transform the MaxSAT instance according to the detected conflicts to compute a tight upper bound. Algorithm 3 formally describes how to apply the δ -Rule and/or (k, δ) -Rule to split LW soft clauses to obtain disjoint inconsistent subsets of soft clauses in MWCLQ. The upper bound computed by Algorithm 3 is very tight, as shown by our experimental results.

Given a weighted graph G , Algorithm 3 first encodes the MWC instance into an LW MaxSAT instance based on an independent set partition of G . The partitioning procedure works as follows. Vertices are sorted in the decreasing order of their weights (ties are broken in favor of the vertices with higher degree) and are successively inserted into an independent set. Suppose that the current independent sets are I_1, I_2, \dots, I_i (in this order, i is 0 at the beginning of the partitioning process). The current first vertex v is inserted into the first I_j such that v is unconnected to all vertices

Algorithm 3: *overestimate*(G), computing an upper bound for MWC by MaxSAT reasoning

Input: A weighted graph $G=(V, E, w)$.

Output: An upper bound for a maximum weight clique in G .

```

1 begin
2   partition  $G$  into independent sets  $I_1, I_2, \dots, I_i$ ;
3   encode  $G$  into an LW MaxSAT instance  $\phi$  and mark all soft clauses available;
4    $UB \leftarrow \sum_{c \in \phi} w(c)$ ;                               /* the base upper bound */
5   while  $\phi$  contains an available soft clause do
6      $c \leftarrow$  the shortest available soft clause of  $\phi$ ;
7     mark  $c$  unavailable;
8     /* Let  $S$  be a set of soft clauses involved in an inconsistent
9     subset and  $S_{topk}$  be a set of top- $k$  literal failed and top- $k$ 
10    empty clauses */
11     $S \leftarrow \emptyset, S_{topk} \leftarrow \emptyset, k \leftarrow 0$ ;
12    while  $k < length(c)$  do
13      if  $\ell_{k+1}$  is failed because of an empty clause then
14         $S \leftarrow S \cup IS(\ell_{k+1})$ ;
15      else if  $\ell_{k+1}$  is failed because of a Top- $k_t$  empty clause  $t$  and  $w_{k_t}(t) > 0$  then
16         $S \leftarrow S \cup (IS(\ell_{k+1}) \setminus \{t\})$ ;
17         $S_{topk} \leftarrow S_{topk} \cup \{t\}$ ;
18      else break;
19       $k \leftarrow k + 1$ ;
20    if  $k > 0$  then
21      if  $k = length(c)$  then
22         $S \leftarrow S \cup \{c\}$ ;                               /* all literals in  $c$  are failed */
23      else if  $w_k(c) > 0$  then
24         $S_{topk} \leftarrow S_{topk} \cup \{c\}$ ;                 /*  $c$  is top- $k$  literal failed */
25      else continue;
26       $\delta \leftarrow \min(w(S), \min_{t \in S_{topk}}(w_{k_t}(t)))$ ;
27       $UB \leftarrow UB - \delta$ ;                               /* improve the upper bound by  $\delta > 0$  */
28       $S'' \leftarrow \emptyset, S''_{topk} \leftarrow \emptyset$ ;
29      foreach clause  $cl \in S$  do
30        apply  $\delta$ -Rule to  $cl$ ;
31         $S'' = S'' \cup \{cl''\}$ ;
32      foreach Top- $k$  literal failed clause or Top- $k$  empty clause  $t \in S_{topk}$  do
33        apply  $(\delta, k_t)$ -Rule to  $t$ ;
34         $S''_{topk} = S''_{topk} \cup \{t''\}$ ;
35       $\phi \leftarrow (\phi \setminus (S \cup S_{topk})) \cup S'' \cup S''_{topk}$ ; /* update soft clauses of  $\phi$  */
36      mark all clauses of  $\phi$  available;
37  return  $UB$ ;                                             /* the improved upper bound by MaxSAT reasoning */

```

already in I_j . If such a I_j does not exist, a new independent set I_{i+1} is opened and v is inserted into I_{i+1} . Each independent set is encoded into an LW soft clause. The total weights of soft clauses is an initial upper bound for MWC of G that should be improved by detecting disjoint inconsistent subsets of soft clauses using the extended MaxSAT reasoning.

The detection of an inconsistent subset of soft clauses is performed by detecting failed literals in the shortest available soft clause c . A literal is failed either because unit propagation falsifies all literals of another clause or just the most weighted literals of another soft clause. In the case all literals of c are failed because unit propagation falsifies all literals of another clause, a usual inconsistent subset of soft clauses is obtained. In the case only the k most weighted literals of c are falsified and/or unit propagation of a literal of c just falsifies the most weighted literals of another soft clause t , the (k, δ) -Rule should be applied to split c and/or t to obtain an inconsistent subset of soft clauses. That is the reason why the *Top-k* empty clauses and *Top-k* literal failed clauses are collected in S_{topk} . Meanwhile, the soft clauses involved in the inconsistent subset are collected in S . In any case, δ is computed as $\delta = \min(w(S), \min_{t \in S_{topk}}(w_{kt}(t)))$ (Line 23), where $w(S)$ is the minimum clause weight in S . Note that S_{topk} is empty when S is a usual inconsistent subset. The δ -Rule and (k, δ) -Rule are applied to split the clauses in the subset and the upper bound is improved by the weight of this subset.

Observe that the detected inconsistent subsets of soft clauses are disjoint, because a soft clause in one subset is not used to detect any other subset. The initial upper bound is improved by the total weight of these inconsistent subsets. Also observe that Algorithm 3 is called at every search tree node from scratch because the graph at different search tree node is different.

4. Empirical Evaluation

In this section, we empirically evaluate MWCLQ and the LW encoding using standard benchmarks, namely the DIMACS benchmark, the BHOSLIB benchmark, random graphs, and the benchmark from the winner determination problem (WDP). We conducted four experiments in this study. The first experiment is to evaluate the performance of MWCLQ by comparing it with other state-of-the-art exact algorithms. The second experiment is to compare different encodings from MWC into MaxSAT. The third experiment is to investigate the impact of splitting soft clauses in the extended MaxSAT reasoning. In the fourth experiment, we applied MWC algorithms to solve WDP instances and compare their performances.

We first introduce the benchmarks used in our experiments and describe the experimental environment. Then we present and discuss the experimental results in detail.

4.1 Benchmarks and Experimental Environment

Three types of benchmarks are used in our first three experiments.

1. **DIMACS** The DIMACS benchmark¹ is taken from the Second DIMACS Implementation Challenge, which has been used widely for benchmarking purposes in the literature of algorithms for MC, MWC, MVC, MIS and so on. The 80 DIMACS instances are generated from real-world applications such as coding theory, fault diagnosis, Keller’s conjecture and the Steiner Triple Problem, as well as random graphs generated with different properties, such

1. <http://www.cs.hbg.psu.edu/txn131/cliique.html>

as the DSJC, brock and p-hat families. The size of these instances ranges from less than 50 vertices and 1,000 edges to more than 4,000 vertices and 5,000,000 edges.

2. **BHOSLIB** BHOSLIB (Benchmarks with Hidden Optimum Solutions for Graph Problems)² instances are based on a CSP model named RB (Xu & Li, 2000; Xu, Boussemart, Hemery, & Lecoutre, 2007). Phase transitions exist for model RB and the transition points can be located exactly. BHOSLIB instances are generated in the phase transition region of model RB and appear to be extremely hard to solve for various algorithms, even when the graph size is small (Liu, Lin, Wang, Su, & Xu, 2011; Xu & Li, 2006). They were firstly used in the SAT competition 2004, and then have been widely used to evaluate algorithms for MC, MVC and MIS.
3. **Random** A random graph of n vertices and density p is generated by randomly selecting each edge with probability p from the complete graph of n vertices. In our experiments, n ranges from 150 to 700 and p from 0.5 to 0.95. Random graphs allow to show the asymptotic behavior of an algorithm.

We convert a non-weighted graph into a weighted graph by associating a weight $w(v_i) = i \bmod 200 + 1$ to each vertex v_i . This method was initially proposed in (Pullan, 2008) and has been used as a standard converting approach to generate weighted graphs from non-weighted instances (Wu et al., 2012; Benlic & Hao, 2013).

All solvers used in our experiments are implemented in C/C++. We compile them using `gcc/g++ 4.7.2` with option `"-O3"`. All experiments are running on a machine with Intel(R) Xeon(R) CPU E5-2680 @ 2.70GHz, 8 cores and 16G RAM under Debian GNU/Linux 7.4. The cut-off time for a solver to solve an instance is one hour (3600 seconds).

4.2 Comparison of MWCLQ with Other Algorithms

We compare MWCLQ with two state-of-the-art exact solvers specific for MWC, a MinSAT solver and CPLEX.

1. Cliquer is a state-of-the-art solver for both MC problem and MWC problem. To our best knowledge, almost all recent exact algorithms for MWC (e.g. (Kumlander, 2004; Shimizu et al., 2013, 2012)) are based on Cliquer. We used the latest version of Cliquer released in 2010, available at its homepage³.
2. DKum is presented in (Kumlander, 2004, 2008), which is implemented in VB. The source code can be found at the author's homepage⁴. To execute it in our experimental environment, we translated it into C.
3. MinSatz (Li et al., 2012) is an exact weighted partial MinSAT solver, which achieves the state-of-the-art performance on solving clique problems and combinatorial auction problems.

2. <http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm>

3. <http://users.tkk.fi/pat/cliquer.html>

4. <http://www.kumlander.eu/graph/index.html>

4. CPLEX⁵ is a high-performance mathematical programming solver for linear programming, mixed integer programming, quadratic programming, and quadratically constrained programming problems. Let $G = (V, E, w)$ be a weighted graph, where $V = \{v_1, v_2, \dots, v_n\}$, the integer programming formulation of the MWC instance G is usually defined as follows.

$$\max \sum_{i=1}^n x_i * w(v_i)$$

subject to

$$\begin{aligned} x_i + x_j &\leq 1, & \forall \{v_i, v_j\} \notin E \\ x_i &\in \{0, 1\}, & i = 1, 2, \dots, n. \end{aligned}$$

Observe that the at-most-one constraint is enforced for every independent set of G . A solution of the integer programming problem corresponds to a maximum weight clique of G . CPLEX 12.6 was used in our experiments to solve MWC after encoding it as an integer programming problem in this way.

Table 1 shows runtimes of different solvers on DIMACS and BHOSLIB benchmarks. All 80 instances from the DIMACS benchmark are used in this experiment. MWCLQ solves 61 instances within the cut-off time, while Cliquer, DKum, MinSatz and CPLEX solves 52, 49, 58 and 44 instances, respectively. For simplicity, we exclude the instances solved within 100 seconds by all solvers or not solved by any solver within 3600 seconds. For the 39 instances displayed in Table 1, MWCLQ outperforms Cliquer on 32 instances and is comparable with Cliquer on other instances. MWCLQ significantly outperforms DKum on all instances except *hamming10-2*, which can be solved by both within 20 seconds. MWCLQ dominates MinSatz on 32 instances. In particular, for the 8 instances from the *brock* family, MWCLQ is about 20X faster than MinSatz for the instances with 400 vertices and solves all four instances with 800 vertices, which cannot be solved by MinSatz. MWCLQ outperforms CPLEX on 28 instances. Particularly, CPLEX cannot solve any instance from *brock* and *p_hat* families, while MWCLQ can solve all of them efficiently. It appears that CPLEX is more effective on *gen* and *MANN* families. It is interesting that both MinSatz and CPLEX solve *MANN_a27* and *MANN_a45* within the cut-off time, which cannot be solved by any specific MWC solver. More interestingly, *MANN_a27* and *MANN_a45* are also hard for heuristic algorithms. For instance, BLS (Benlic & Hao, 2013) can only find an approximating solution of 12281 with the success rate 16% within 396.58 seconds for *MANN_a27*, while PLS (Pullan, 2008) can only find an approximating solution of 12264.

We also used all 40 instances from the BHOSLIB benchmark. These instances are extremely hard for exact MWC solvers. All evaluated solvers can solve at most the 5 smallest instances with 450 vertices. Both MWCLQ and DKum solve 5 instances, CPLEX, Cliquer and MinSatz solves 3, 1 and 0 instances, respectively. Moreover, MWCLQ achieves the best performance on 4 out of the 5 instances.

Table 2 shows mean runtimes on random graphs. We generate 50 graphs at each point. MWCLQ is the only algorithm which solves all 700 graphs in this experiment within the cutoff time, while Cliquer, Dkum, MinSatz and CPLEX solves 595, 548, 548 and 387 instances, respectively.

5. <http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

Table 1: Runtimes (in seconds) on DIMACS and BHOSLIB benchmarks. The cut-off time is 3600 seconds. $|V|$ stands for the number of vertices, D for the density of the graph and ω_v for the optimal solution of MWC. When a solver cannot solve an instance within 3600 seconds, its runtime is marked by '-'. Instances solved within 100 seconds by all solvers or not solved by any solver within the cut-off time are omitted.

Graph	$ V $	D	ω_v	Cliquer	DKum	MinSatz	CPLEX	MWCLQ
brock400_1	400	74	3422	260.2	607.2	2046	-	129.3
brock400_2	400	75	3350	366.1	524.5	2737	-	127.7
brock400_3	400	74	3471	290.7	573.6	2363	-	107.2
brock400_4	400	75	3626	287.4	606.3	1609	-	81.81
brock800_1	800	65	3121	1548	-	-	-	1427
brock800_2	800	65	3043	1603	-	-	-	2002
brock800_3	800	65	3076	1702	-	-	-	1545
brock800_4	800	65	2971	1990	-	-	-	2039
C250.9	250	89	5092	-	2215	640.6	30.41	39.99
DSJC1000.5	1000	50	2186	32.50	91.88	3527	-	81.30
DSJC500.5	500	50	1725	0.97	0.97	32.80	-	0.92
gen200_p0.9_44	200	89	5043	678.6	119.8	72.40	2.44	7.00
gen200_p0.9_55	200	89	5416	1718	362.2	35.19	1.81	2.76
gen400_p0.9_75	400	90	8006	-	-	-	238.7	-
hamming10-2	1024	99	50512	1469	7.47	-	0.06	15.46
johnson32-2-4	496	88	2033	-	-	-	0.43	-
MANN_a27	378	98	12283	-	-	4.45	2.07	-
MANN_a45	1035	99	34265	-	-	3487	32.61	-
p_hat1000-1	1000	24	1514	0.12	0.35	21.75	-	0.53
p_hat1000-2	1000	49	5777	-	-	1628	-	2501
p_hat1500-1	1500	25	1619	1.03	2.72	252.3	-	4.12
p_hat500-1	500	25	1231	0.02	0.02	1.52	-	0.01
p_hat500-2	500	50	3920	5.59	3.86	10.41	-	2.34
p_hat500-3	500	75	5375	-	-	1009	-	916.7
p_hat700-1	700	25	1441	1.03	0.17	4.42	-	0.13
p_hat700-2	700	50	5290	169.5	213.4	39.93	-	47.88
san1000	1000	50	1716	-	-	46.90	-	183.8
san200_0.7_2	200	69	2422	403.9	23.28	0.11	0.70	0
san200_0.9_1	200	89	6825	-	166.9	0.10	0.31	0.24
san200_0.9_2	200	89	6082	223.3	15.11	16.37	0.83	1.64
san200_0.9_3	200	89	4748	2329	470.3	118.2	3.54	16.19
san400_0.7_1	400	70	3941	-	-	12.62	24.45	3.44
san400_0.7_2	400	70	3110	-	-	34.74	87.75	4.98
san400_0.7_3	400	70	2771	-	-	96.12	43.53	6.37
san400_0.9_1	400	90	9776	-	-	1849	13.30	1257
sanr200_0.7	200	69	2325	0.20	-	2.73	10.81	0.13
sanr200_0.9	200	89	5126	1150	215.3	68.97	11.01	6.40
sanr400_0.5	400	50	1835	0.15	-	9.44	-	0.31
sanr400_0.7	400	70	2992	29.98	74.48	429.0	-	25.57
Total: 80				52	49	58	44	61
rb30-15-1	450	82	2990	-	177.7	-	-	244.9
frb30-15-2	450	82	3006	1065	44.98	-	-	30.14
frb30-15-3	450	82	2995	-	423.4	-	193.5	131.7
frb30-15-4	450	82	3032	-	287.5	-	258.2	181.8
frb30-15-5	450	82	3011	-	154.0	-	118.3	57.31
Total: 40				1	5	0	3	5

Table 2: Mean runtimes in seconds on random graphs, obtained by solving 50 graphs at each point. The cut-off time is 3600 seconds. $|V|$ stands for the number of vertices, D for the density of the graph and \bar{w}_v for the weight of an optimal solution, averaged over 50 graphs. The mean runtime is marked by '-' if no instance is solved at the point within the cut-off time. # stands for the number of instances solved by the solver within the cut-off time

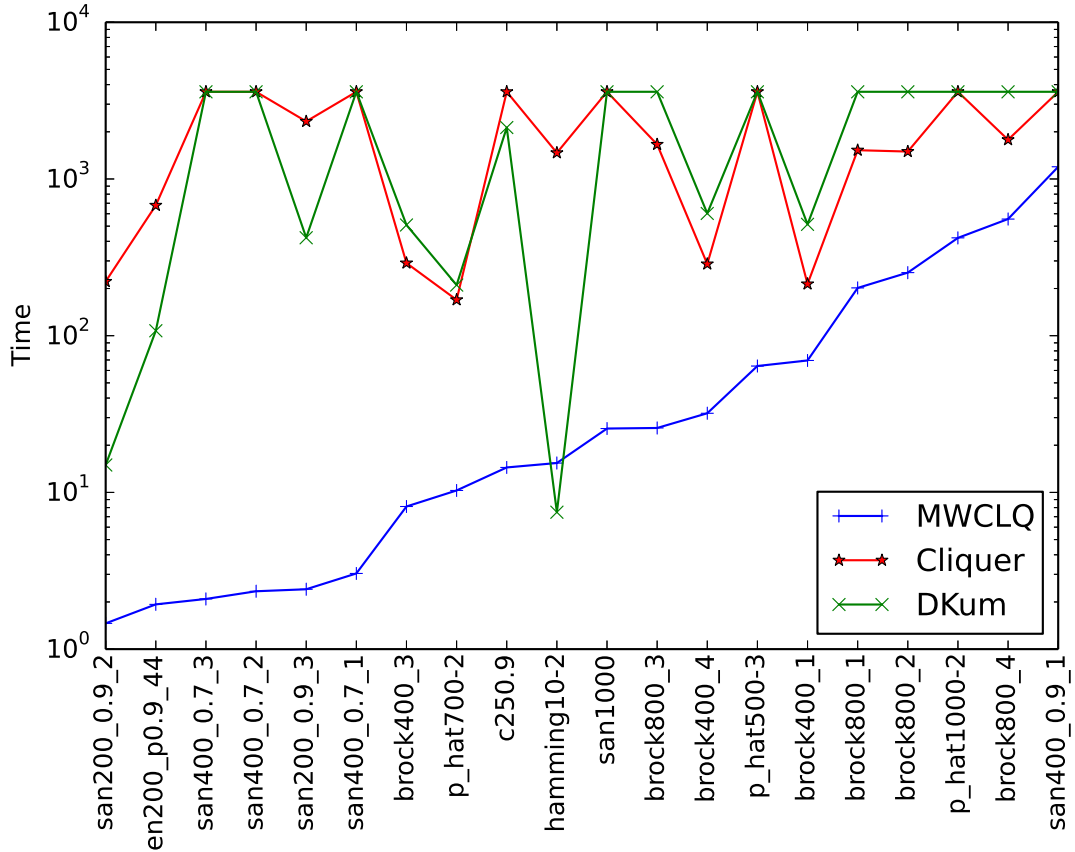
Benchmark			Cliquer		DKum		MinSatz		CPLEX		MWCLQ	
$ V $	D	\bar{w}_v	Time	#	Time	#	Time	#	Time	#	Time	#
150	0.90	3394	21.06	50	7.92	50	3.57	50	2.58	50	0.58	50
150	0.95	4766	1006	50	35.42	50	2.21	50	1.99	50	0.42	50
200	0.80	3249	4.02	50	5.05	50	12.65	50	54.83	50	0.99	50
200	0.90	5095	974.9	50	373.0	50	94.63	50	23.45	50	14.30	50
200	0.95	7372	-	0	2464	19	111.1	50	3.91	50	28.77	50
300	0.70	2441	1.76	50	3.01	50	32.36	50	353.5	50	1.13	50
300	0.80	3334	81.45	50	107.1	50	379.6	50	303.7	47	14.87	50
300	0.90	5351	-	0	-	0	-	0	263.5	40	845.4	50
500	0.60	2285	3.94	50	9.45	50	208.5	50	-	0	6.03	50
500	0.70	2969	119.9	50	302.1	50	2994	48	-	0	93.36	50
600	0.60	2496	23.51	50	62.46	50	1006	50	-	0	34.79	50
600	0.70	3293	1256	50	2884	29	-	0	-	0	869.9	50
700	0.60	2510	46.01	50	132.8	50	2757	50	-	0	78.24	50
700	0.70	3283	3006	45	-	0	-	0	-	0	2434	50
Total 700			595		548		548		387		700	

MWCLQ significantly outperforms Dkum and MinSatz on all instances and dominates CPLEX at all points except (200, 0.95) where both algorithms solve all instances within 30 seconds. MWCLQ dominates Cliquer completely on random graphs with density $D \geq 0.7$, and is comparable with Cliquer when $D < 0.7$. MWCLQ is the only solver which can solve all instances at the point (700, 0.7). The experimental results suggest in particular that MWCLQ is very effective in solving graphs with high density. CPLEX is also effective for dense graphs, but it cannot solve any random instance with more than 500 vertices.

Table 3: Lower bounds given by different solvers for DIMACS instances which cannot be solved by any solver within the cut-off time.

Instance	$ V $	D	Cliquer	DKum	MinSatz	CPLEX	MWCLQ
C1000.9	1000	90	1181	1846	6385	8066	8471
C2000.5	2000	50	2466	1186	2198	1358	2466
C2000.9	2000	90	534	782	6747	7362	10034
C4000.5	4000	50	1284	863	2263	1854	2698
C500.9	500	90	1868	2070	5594	6520	6672
MANN_a81	3321	100	195	1201	100970	111386	111033
gen400_p0.9_55	400	90	2501	3037	5988	6611	6676
gen400_p0.9_65	400	90	2855	3179	6180	6720	6832
hamming10-4	1024	83	738	1798	4062	5042	4614
keller5	776	75	2860	2139	3317	3317	3317
keller6	3361	82	511	1637	5595	6937	6316
p_hat1000-3	1000	74	2417	2893	6779	7258	7588
p_hat1500-2	1500	51	2897	3127	6100	5954	7104
p_hat1500-3	1500	75	1521	2067	7050	9058	8449
p_hat700-3	700	75	2822	5156	7340	7400	7565

Figure 4: Runtimes to find an optimal solution. Instances for which all solvers find an optimal solution within 100 seconds or MWCLQ finds an optimal solution in less than one second are omitted



For the 15 DIMACS instances cannot be solved by any algorithm within the cut-off time, we report the largest weight clique found by each solver before the algorithm terminates, which is a lower bound of the optimal solution. Table 3 shows MWCLQ computes the best lower bounds for 11 instances, while CPLEX gives the best lower bounds for 5 instances. MWCLQ can give a significantly better lower bound than other MWC solvers on all instances except *C2000.5*, where MWCLQ and Cliquer share the same bound. This result suggests that MWCLQ can often compute a better approximate solution than other exact solvers within a given time.

An exact algorithm for MC or MWC usually solves an instance in two phases. In the first phase, the algorithm finds an optimal solution. Then in the second phase, the algorithm proves that the solution is indeed optimal by showing that no better solution exists. In Fig. 4, we compare the runtimes that Cliquer, Dkum and MWCLQ need to find an optimal solution of a DIMACS instance. For simplicity, the instances for which all solvers find an optimal solution within 100 seconds or MWCLQ finds an optimal solution in less than one second are omitted. MWCLQ always finds an

optimal solution much faster than other solvers on all instances except *hamming10-2*. For a few instances such as *brock800_2*, *brock800_4*, although MWCLQ spends more time than Cliquer for exact solving them (Table1), it finds an optimal solution 10 times faster than Cliquer.

In summary, both Table 3 and Fig. 4 show that MWCLQ generally finds an optimal solution much faster than other solvers, although sometimes it may spend more time to prove the optimality.

4.3 Effectiveness of the Literal-Weighted Encoding

We presented four encodings from MWC into MaxSAT, namely the direct encoding, the split encoding, the iterative split encoding and the LW encoding. When an MWC instance is encoded into a MaxSAT instance using the first three encodings, a MaxSAT solver can be used to search for an optimal solution of the MWC instance. However, if the MWC instance is encoded into a MaxSAT instance using the LW encoding, the optimal solution found by the MaxSAT solver usually does not correspond to a maximum weight clique (see Example 5). Therefore, when a MaxSAT solver is used to solve an MWC instance, only the first three encodings can be used.

Different from MaxSAT solvers, MaxSAT reasoning in MWCLQ is just used in the upper bounding procedure at each search tree node, where the current subgraph is dynamically encoded into a MaxSAT instance to apply MaxSAT reasoning to detect inconsistent subsets of soft clauses. Recall that MaxSAT reasoning in MWCLQ is based on the LW encoding from MWC into MaxSAT.

We implemented three other versions of MWCLQ, i.e., $MWCLQ_{dir}$, $MWCLQ_{sp}$ and $MWCLQ_{it}$, which are identical to MWCLQ, but are based on the direct encoding, the split encoding and the iterative split encoding, respectively. In a MaxSAT instance obtained using the direct encoding, all inconsistent subsets only contain two soft clauses, because all soft clauses are unit. When $MWCLQ_{dir}$ detects an inconsistent subset such as $\{(x_1, w(x_1)), (x_2, w(x_2))\}$, the most weighted clause in the subset, say, $(x_2, w(x_2))$, is split into $(x_2, w(x_1))$ and $(x_2, w(x_2) - w(x_1))$. Then the upper bound is improved by $w(x_1)$, and the clause $(x_2, w(x_2) - w(x_1))$ can be used for further detection. In a MaxSAT instance obtained using the split and the iterative split encoding, all vertices in the same soft clause have the same weight. Observe that the *Top-k* literal failed clause and the *Top-k* empty clause do not make sense, and the (δ, k) -Rule is not needed in the three encoding.

In this experiment, we ran the following four different kinds of state-of-the-art MaxSAT solvers to solve the MWC instances encoded into MaxSAT using the direct encoding, the split encoding and the iterative split encoding.

1. akMaxSat (Kügel, 2010) is a branch-and-bound MaxSAT solver that computes the lower bound with a combination of MaxSAT resolution and detection of disjoint inconsistent subsets. One of its authors sent us the source code submitted to the MaxSAT evaluation 2012.
2. MaxSatz (Li et al., 2007) is a branch-and-bound MaxSAT solver that incorporates some SAT technologies. We used the latest version MaxSatz2013, which is one of the best solvers in the MaxSAT evaluation 2013.
3. WPM1-2013 (Ansótegui et al., 2009) is a MaxSAT solver based on successive calls to a SAT solvers. One of its authors provided us an executable file of WPM which is used in the MaxSAT evaluation 2013.
4. MaxHS (Davies & Bacchus, 2013) is a hybrid Maxsat solver that exploits both SAT and integer programming technologies. It is available at <http://www.maxhs.org/>.

We report the number of instances solved by akMaxSat, MaxSatz, MaxHS, MWCLQ_{dir}, MWCLQ_{sp}, MWCLQ_{it} and MWCLQ within 3600 seconds. The results of WPM are not reported, because it solves only 3 DIMACS instances and does not solve any BHOSLIB instance. In addition, it always runs out of memory when solving random instances.

Table 4: Number of instances solved within 3600 seconds by MaxSAT solvers using three different encodings and four versions of MWCLQ. Direct stands for the direct encoding, Split is for the split encoding, Iter is for the iterative split encoding and LW is for the literal-weighted encoding.

Instance		akMaxSat			MaxSatz			MaxHS			MWC-	MWC-	MWC-	MWC-
Benchmark	#	Direct	Split	Iter	Direct	Split	Iter	Direct	Split	Iter	LQ _{dir}	LQ _{sp}	LQ _{it}	LQ
brock	12	4	4	4	4	4	4	4	0	0	4	8	8	12
c-fat	7	7	7	7	7	7	7	7	7	7	7	7	7	7
C	7	2	2	2	2	2	2	2	1	1	1	2	2	2
DSJC	2	0	0	0	1	1	1	0	0	0	2	2	2	2
gen	5	2	2	2	2	2	2	3	0	0	0	2	2	2
hamming	6	5	5	5	5	5	5	5	4	3	3	4	4	5
johnson	4	3	3	3	3	3	3	4	3	2	3	3	3	3
keller	3	1	1	1	1	1	1	1	0	0	1	1	1	1
MANN	4	1	1	1	2	2	2	3	2	2	1	1	1	1
p_hat	15	5	4	5	7	5	7	4	1	0	8	8	9	11
san	15	7	9	9	8	8	8	13	4	3	6	10	14	15
DIMACS: 80		37	38	39	42	40	42	46	22	18	36	48	53	61
BHOSLIB: 40		0	0	1	0	0	5	3	0	0	0	0	5	5
(150,0.9)	50	43	50	50	50	50	50	50	7	7	50	50	50	50
(150,0.95)	50	46	50	50	49	50	50	50	50	50	0	50	50	50
(200,0.8)	50	38	50	50	49	50	50	50	0	0	50	50	50	50
(200,0.9)	50	44	50	50	49	50	50	50	0	0	0	41	50	50
(200,0.95)	50	44	50	50	49	50	50	50	14	14	0	25	50	50
(300,0.7)	50	30	48	50	50	50	50	50	0	0	50	50	50	50
(300,0.8)	50	14	7	9	47	19	18	41	0	0	50	50	50	50
(300,0.9)	50	8	8	8	2	0	0	35	0	0	0	0	29	50
(500,0.6)	50	0	0	0	0	0	0	0	0	0	50	50	50	50
(500,0.7)	50	0	0	0	0	0	0	0	0	0	49	41	50	50
(600,0.6)	50	0	0	0	0	0	0	0	0	0	50	50	50	50
(600,0.7)	50	0	0	0	0	0	0	0	0	0	0	0	39	50
(700,0.6)	50	0	0	0	0	0	0	0	0	0	50	50	50	50
(700,0.7)	50	0	0	0	1	0	0	0	0	0	0	0	0	50
RAND: 700		267	313	317	346	319	318	376	71	71	399	507	618	700

Experimental results in Table 4 suggest that encoding approaches do affect the performances of MaxSAT solvers to solve an MWC instance. However, the effectiveness of the direct encoding, the split encoding and the iterative encoding for different MaxSAT solvers are not clear. Concretely, the iterative split encoding makes akMaxSat a little faster than other encodings do. MaxHS using the direct encoding dominates MaxHS using other encodings. The iterative split encoding is a better choice for MaxSatz to solve BHOSLIB instances, but the direct encoding is better to solve random graphs. These results also show that MWCLQ significantly outperforms MaxSAT solvers to solve an MWC instance. We observe that although MaxSAT reasoning is powerful in improving the upper bound in a BnB algorithm for MC or MWC, using a MaxSAT solver to solve an MWC instance is not effective.

Thanks to the LW encoding, MWCLQ significantly outperforms MWCLQ_{dir}, MWCLQ_{sp} and MWCLQ_{it}. MWCLQ_{dir} and MWCLQ_{sp} are slow in solving the MWC instances because they

cannot capture the graph structure well. Although the iterative split encoding in $MWCLQ_{it}$ can capture the graph structure, $MWCLQ_{it}$ does not exploit the power of MaxSAT reasoning efficiently, because the clause splittings in $MWCLQ_{it}$ are not guided by conflicts.

The advantages of MWCLQ can be described as follows: (1) it is a BnB algorithm for MWC that is able to exploit the graph structure, especially by ordering vertices and by partitioning the graph into independent sets at each search tree node, that a MaxSAT solver does not do; (2) it exploits the power of MaxSAT reasoning in the upper bounding procedure to derive a tight upper bound, that a classical BnB algorithm for MWC does not do. Nevertheless, to make MaxSAT reasoning beneficial, a relevant encoding from MWC into MaxSAT is necessary, as shown in Table 4. The encoding has an obvious impact on the performance of MWCLQ.

4.4 Effectiveness of the Upper Bound Based on MaxSAT reasoning

Table 5: Runtimes (in seconds) and search tree sizes (in thousands) of different versions of MWCLQ for solving the DIMACS and BHOSLIB instances, as well as the gain ratio of MWCLQ and MWCLQ- compared with MWCLQ--. The cut-off time is 3600 seconds. '-' means that the solver cannot solve the instance within the cut-off time. Instances solved within 10 seconds by all solvers or not solved by any within the cut-off time are omitted.

Benchmark	MWCLQ--		MWCLQ-				MWCLQ			
	Time	Size	Time	Gain	Size	Gain	Time	Gain	Size	Gain
brock400_1	209.0	63153	234.3	0.89	54505	1.16	129.3	1.62	25367	2.49
brock400_2	203.5	58770	227.7	0.89	51244	1.15	127.7	1.59	23788	2.47
brock400_3	173.4	50079	194.2	0.89	43616	1.15	107.2	1.62	19830	2.53
brock400_4	131.5	37420	146.3	0.90	32853	1.14	81.81	1.61	15133	2.47
brock800_1	1813	421259	1990	0.91	396425	1.06	1427	1.27	223715	1.88
brock800_2	2563	648486	2695	0.95	605433	1.07	2002	1.28	339910	1.91
brock800_3	1968	470594	2201	0.89	442518	1.06	1545	1.27	248218	1.90
brock800_4	2623	675790	2931	0.89	633367	1.07	2039	1.29	352700	1.92
C250.9	159.4	31542	160.5	0.99	20114	1.57	39.99	3.99	4472	7.05
DSJC1000.5	87.73	26154	91.87	0.95	25207	1.04	81.30	1.08	17032	1.54
gen200_p0.9_44	27.83	7560	23.12	1.20	3692	2.05	7.00	3.98	979	7.72
gen200_p0.9_55	10.41	2530	8.70	1.20	1310	1.93	2.76	3.77	344	7.34
hamming10-2	-	-	-	-	-	-	15.46	-	41758	-
p_hat1000-2	-	-	-	-	-	-	2501	-	176892	-
p_hat500-3	3009	457787	2813	1.07	238949	1.92	916.7	3.28	77115	5.94
p_hat700-2	113.4	17715	117.5	0.97	10571	1.68	47.88	2.37	4186	4.23
san1000	135.9	31173	137.2	0.99	30680	1.02	183.8	0.74	21694	1.44
san200_0.9_3	56.31	15955	48.47	1.16	7946	2.01	16.19	3.48	2180	7.32
san400_0.7_3	9.79	2936	10.53	0.93	2403	1.22	6.37	1.54	1213	2.42
san400_0.9_1	-	-	-	-	-	-	1257	-	63106	-
sanr200_0.9	25.90	6690	20.10	1.29	3001	2.23	6.40	4.05	850	7.87
sanr400_0.7	35.71	12228	39.60	0.90	10797	1.13	25.57	1.40	5705	2.14
frb30-15-1	411.3	156311	394.8	1.04	108476	1.44	244.9	1.68	27928	5.60
frb30-15-2	42.28	13416	39.17	1.08	9287	1.44	30.14	1.40	2597	5.17
frb30-15-3	171.8	67246	167.0	1.03	48401	1.39	131.7	1.30	14589	4.61
frb30-15-4	267.9	83139	257.3	1.04	63411	1.31	181.8	1.47	16336	5.09
frb30-15-5	85.27	22362	82.23	1.04	17705	1.26	57.31	1.49	5575	4.01

To study the impact of MaxSAT reasoning with soft clause splitting, we implemented two derived versions of MWCLQ, namely, MWCLQ-- and MWCLQ-. MWCLQ-- is identical to MWCLQ except that it only uses the trivial bound based on the independent partition of the graph, which is equal to the sum of the largest weight in each independent set. MWCLQ- is identical to MWCLQ

Table 6: Runtimes (in seconds) and search tree sizes (in thousands) of different versions of MWCLQ for the random instances, averaged by solving 50 instances at each point, as well as the gain ratio of MWCLQ and MWCLQ- compared with MWCLQ-. The cut-off time is 3600 seconds for a solver to solve one instance.

Graph		MWCLQ--			MWCLQ-					MWCLQ				
$ V $	D	#	Time	Size	#	Time	Gain	Size	Gain	#	Time	Gain	Size	Gain
150	0.90	50	1.85	715	50	1.34	1.38	255	2.81	50	0.58	3.19	108	6.58
150	0.95	50	5.23	1440	50	1.28	4.09	123	11.69	50	0.42	12.45	46	30.69
200	0.80	50	1.71	725	50	1.89	0.90	539	1.34	50	0.99	1.73	245	2.96
200	0.90	50	49.41	14030	50	40.13	1.23	6109	2.30	50	14.30	3.46	2031	6.91
200	0.95	50	405.8	81772	50	117.1	3.47	8583	9.53	50	28.77	14.10	2454	33.32
300	0.70	50	1.58	552	50	1.80	0.88	493	1.12	50	1.13	1.40	263	2.10
300	0.80	50	27.09	7655	50	30.98	0.87	6265	1.22	50	14.87	1.82	2611	2.93
300	0.90	31	2261	399832	33	2261	1.00	245004	1.63	50	845.4	2.67	85676	4.67
500	0.60	50	7.53	2310	50	8.12	0.93	2176	1.06	50	6.03	1.25	1314	1.76
500	0.70	50	133.8	34848	50	149.5	0.89	31905	1.09	50	93.36	1.43	16326	2.13
600	0.60	50	42.85	13483	50	46.40	0.92	12661	1.06	50	34.79	1.23	7667	1.76
600	0.70	50	1229	320607	50	1390	0.88	292627	1.10	50	869.9	1.41	149522	2.14
700	0.60	50	94.10	24897	50	104.8	0.90	23624	1.05	50	78.24	1.20	14219	1.75
700	0.70	31	3177	682683	15	3343	0.95	575774	1.19	50	2434	1.31	351894	1.94

except it only detects the inconsistent subsets of soft clauses, but does not use the δ -Rule and the (δ, k) -Rule to split clauses in the subset.

Table 5 shows runtimes (in seconds) and search tree sizes (in thousands) of MWCLQ--, MWCLQ- and MWCLQ for solving the DIMACS and BHOSLIB instances, as well as the gain ratio of MWCLQ- and MWCLQ compared with MWCLQ-- in terms of runtime and search tree size computed as MWCLQ--/MWCLQ- and MWCLQ--/MWCLQ, respectively. MaxSAT reasoning without soft clause splitting in MWCLQ- makes MWCLQ- better than MWCLQ-- in terms of search tree size. But the reduction of search tree size is not enough in general to compensate the overhead of MaxSAT reasoning, so that the gain of MWCLQ- compared with MWCLQ-- in terms of runtime is not clear. However, the soft clause splitting using the δ -Rule and the (δ, k) -Rule allows MaxSAT reasoning to prune much more search space, making MWCLQ substantially (from 1.08 to 4.05 times) faster than MWCLQ-. In fact, MWCLQ solves 3 instances more than MWCLQ-- and MWCLQ-, and significantly outperforms them on all instances except *san1000*. Observe that MWCLQ-- and MWCLQ- solve the same number of instances as Minsatz (58) on the DIMACS benchmark, and solve more instances than Cliquer (52), Dkum (49) and CPLEX (44).

Table 6 shows mean runtimes (in seconds) and mean search tree sizes (in thousands) of different versions of MWCLQ for the random instances, averaged by solving 50 instances at each point, as well as the gain ratio of MWCLQ and MWCLQ- compared with MWCLQ-. MWCLQ solves all instances. However, neither MWCLQ-- nor MWCLQ- can solve all graphs at the points (300, 0.90) and (700, 0.7). MaxSAT reasoning allows MWCLQ- to prune more search space than MWCLQ-- on all instances, but the overhead makes MWCLQ- slower on instances with more than 300 vertices. However, similarly as in DIMACS and BHOSLIB cases, MWCLQ is from 1.2 to 14.3 times faster than MWCLQ--, because the soft clause splittings using the δ -Rule and the (δ, k) -Rule allow MWCLQ to substantially reduce the search space on all instances. Furthermore, the gain in terms of both runtime and search tree size increases with the density of graph.

4.5 Application of MWCLQ for the Winner Determination Problem

An important application of MWC is to solve the optimal winner determination problem (WDP) in combinatorial auctions. The auctioneer has a set of m items, $M = \{1, 2, \dots, m\}$, to sell, and the buyers submit a set of n bids, $B = \{B_1, B_2, \dots, B_n\}$. A bid is a pair $B_i = (S_i, P_i)$, where $S_i \subset M$ is a subset of items and $P_i \geq 0$ is a price for all items in S_i . The WDP problem is to determine the bids as winning or losing so as to maximum the auctioneer’s revenue, such that each item can be allocated to at most one bidder. We define an MWC instance $G = (V, E, w)$ for the WDP instance as follows.

- For each bid $B_i \in B$, define a vertex v_i with weight $w(v_i) = P_i$, i.e., $V = \{v_1, v_2, \dots, v_n\}$ and $w(v_i) = P_i$;
- Add an edge $\{v_i, v_j\}$ to G if and only if B_i and B_j do not share common item, i.e., $E = \{\{v_i, v_j\} \mid S_i \cap S_j = \emptyset, 1 \leq i < j \leq n\}$.

A maximum weight clique of G corresponds to a feasible subset of bids with a maximum revenue.

In this experiment, we compared MWCLQ with Cliquer, Dkum, MinSatz and CPLEX on realistic WDP instances. We also selected MaxHS using the direct encoding, which is the most effective MaxSAT solver to solve an MWC instance, in this comparison. We used the benchmark provided by Lau and Goh (Lau & Goh, 2002), which has been widely used as benchmark purpose to test WDP algorithms (Guo, Lim, Rodrigues, & Zhu, 2006; Sghir, Hao, Jaafar, & Ghédira, 2014; Wu & Hao, 2015). Instances in this benchmark are generated by incorporating the following factors, i.e., a pricing factor which models a bidder’s acceptable price range for each bid, a preference factor which takes into account bidder’s preferences among bids, and a fairness factor which measures the fairness in distributing items among bidders. The benchmark contains 500 instances with up to 1500 item and 1500 bids, which can be divided into 5 groups by the item number and the bid number. Each group contains 100 instances labeled as *REL- m - n* , where m is the number of items and n is the number of bids.

Table 7: Mean runtimes in seconds on WDP instances, obtained by solving for 100 graphs of each group. The cut-off time is 3600 seconds. The mean runtime is marked by ‘-’ if no instance is solved in the group within the cut-off time. # stands for the number of instances solved by the solver

Benchmark		Cliquer		DKum		MinSatz		MaxHS		CPLEX		MWCLQ	
Group	#	Time	#	Time	#	Time	#	Time	#	Time	#	Time	#
REL-500-1000	100	809.9	100	628.3	100	552.5	100	-	0	-	0	55.61	100
REL-1000-1000	100	3.73	100	4.52	100	25.39	100	-	0	-	0	1.45	100
REL-1000-500	100	0.03	100	0.03	100	0.81	100	1003.7	100	266.7	100	0.05	100
REL-1000-1500	100	2.84	100	3.55	100	67.90	100	-	0	-	0	1.56	100
REL-1500-1500	100	3.38	100	5.30	100	78.19	100	-	0	-	0	2.42	100

Table 7 summarizes the mean runtimes and numbers of instances solved within the cut-off time by group. Results show that MWCLQ outperforms other solvers on all groups except *REL-1000-500*, where Cliquer, Dkum, MinSatz and MWCLQ are comparable. MWCLQ achieves at least 10X speedup for the instances in the hardest group *REL-500-1000*. MaxHS and CPLEX are not effective in solving these instances. In our experiment, we transformed WDP to MWC first and then

formulated it with integer programming. Another method is used in (Wu & Hao, 2015) to formulate WDP directly into integer programming. The method in (Wu & Hao, 2015) appears to be slightly more effective than the transformation in our experiment, but it does not affect our comparison, because in (Wu & Hao, 2015), the only instances CPLEX is able to solve within 3600 seconds are also from *REL-1000-500*.

Table 8 reports the detailed comparative results on the first 10 instances from each group. MWCLQ outperforms other solvers on all instances except the 10 instances from the easiest group (in401, in402, ..., in410), which can be solved by Cliquer, DKum, MinSatz and MWCLQ within less than one second. These results suggest that MWCLQ is more effective in solving MWC instances from WDP than other specific MWC algorithms as well as MinSatz, MaxHS and CPLEX. Moreover, MWCLQ is even more efficient than some state-of-the-art heuristic algorithms on some relatively hard instances. For example, MWCLQ solves *in108* in 101.04 seconds, while the heuristic algorithm, based on tabu search, presented in (Wu & Hao, 2015) takes 113.53 seconds to find the solution with probability 0.73. Ignoring the difference of running environments, MWCLQ is faster than the tabu search algorithm on *in108*. Note that heuristic algorithms can only give a feasible solution, but cannot guarantee the optimality.

5. Conclusion

MaxSAT reasoning has been proved to be very effective for the MC problem, based on a partition of the graph into independent sets. However, MaxSAT reasoning can not be naturally extended to solve the MWC problem because of the literal weights, as shown by the relatively poor performance of MWCLQ-, MWCLQ_{dir}, MWCLQ_{sp} and MWCLQ_{it}. MWCLQ- exploits MaxSAT reasoning but does not deal with literal weights. The encodings from MWC into MaxSAT used in MWCLQ_{dir} and MWCLQ_{sp} do not capture well the graph structure. Although MWCLQ_{it} can exploit the graph structure, it does too many useless splits and has difficulties to take advantage of MaxSAT reasoning efficiently. We thus propose to encode an MWC instance into a literal-weighted MaxSAT instance, in which both soft clauses and literals in soft clauses are weighted. At the same time, we define two notions, namely the *Top-k* literal failed clause and the *Top-k* empty clause, to extend the classical MaxSAT reasoning and introduce two clause splitting rules called the δ -Rule and the (δ, k) -Rule to deal with both literal weights and clause weights.

The LW encoding from MWC into MaxSAT is always based on a partition of a graph into independent sets, each soft clause corresponding to an independent set. Usually this independent set partition is far from optimal for improving an upper bound for MWC using MaxSAT reasoning. For this reason, we extend MaxSAT reasoning by identifying the *Top-k* literal failed clause and the *Top-k* empty clause. Meanwhile, we split soft clauses using the δ -Rule and the (δ, k) -Rule, according to the conflicts detected by MaxSAT reasoning, to obtain as many inconsistent subsets as possible. Consequently, we obtain a tight upper bound for MWC to prune search space. This approach is implemented in MWCLQ, which is substantially better than MWCLQ-, MWCLQ_{dir}, MWCLQ_{sp} and MWCLQ_{it}, confirming the effectiveness of the approach. MWCLQ is also favorably compared with the state-of-the-art MWC solvers as Cliquer, DKum, MinSatz and CPLEX, as well as state-of-the-art MaxSAT solvers using different encodings, on standard benchmarks and instances from realistic applications.

Table 8: Runtimes on WDP instances. The cut-off time is 3600 seconds. $|V|$ stands for the number of vertices, D for the density of the graph after transforming WDP into MWC and ω_v for the optimal solution of MWC. The runtime is in seconds. When a solver cannot solve an instance within 3600 seconds, its runtime is marked by '-'. We only show the first 10 instances in each group

Graph	$ V $	D	ω_v	Cliquer	DKum	MinSatz	MaxHS	CPLEX	MWCLQ
in101	1000	0.31	72724.617	819.6	616.1	610.8	-	-	53.87
in102	1000	0.30	72518.222	414.9	289.0	308.8	-	-	34.40
in103	1000	0.31	72129.500	551.4	455.5	479.6	-	-	44.94
in104	1000	0.30	72709.646	330.8	218.1	353.5	-	-	37.00
in105	1000	0.30	75646.127	840.3	547.3	367.7	-	-	37.33
in106	1000	0.30	71258.613	272.7	346.4	270.7	-	-	23.56
in107	1000	0.30	69713.403	595.9	460.1	736.9	-	-	68.07
in108	1000	0.31	75813.205	1321	1121	924.9	-	-	101.04
in109	1000	0.30	69475.895	247.3	238.8	305.0	-	-	29.10
in110	1000	0.30	68295.289	308.5	307.7	411.7	-	-	43.16
in201	1000	0.15	81557.742	1.80	2.47	22.09	-	-	1.01
in202	1000	0.16	90708.127	3.73	4.98	25.90	-	-	1.37
in203	1000	0.16	86239.214	3.63	6.65	26.66	-	-	1.57
in204	1000	0.17	87075.428	6.13	7.20	30.47	-	-	2.01
in205	1000	0.16	86515.951	3.11	4.85	27.42	-	-	1.64
in206	1000	0.16	91518.964	2.37	3.37	22.85	-	-	1.11
in207	1000	0.17	93129.248	3.95	5.69	28.18	-	-	1.69
in208	1000	0.16	94904.679	4.07	5.18	22.09	-	-	1.05
in209	1000	0.16	87268.965	2.13	3.38	26.66	-	-	1.54
in210	1000	0.16	89962.396	3.54	4.78	22.09	-	-	1.21
in401	500	0.15	77417.482	0.02	0.03	0.69	923.01	181.70	0.04
in402	500	0.15	76273.336	0.02	0.02	0.72	1107.2	198.12	0.04
in403	500	0.16	74843.957	0.03	0.04	0.76	891.23	229.0	0.04
in404	500	0.17	78761.690	0.05	0.08	0.89	1082.9	208.97	0.06
in405	500	0.17	75915.900	0.05	0.08	0.88	1300.6	224.98	0.06
in406	500	0.14	72863.324	0.02	0.02	0.75	992.34	175.15	0.05
in407	500	0.17	76365.717	0.05	0.07	0.79	1569.1	364.58	0.06
in408	500	0.16	77018.833	0.04	0.04	0.82	1187.2	199.39	0.06
in409	500	0.14	73188.619	0.01	0.02	0.71	846.56	217.31	0.04
in410	500	0.17	73791.658	0.04	0.06	0.85	1132.1	243.42	0.06
in501	1500	0.09	88656.958	4.64	5.68	81.25	-	-	2.30
in502	1500	0.08	86236.911	1.69	2.30	65.16	-	-	1.45
in503	1500	0.08	87812.377	3.44	4.42	72.06	-	-	1.72
in504	1500	0.08	85600.001	2.37	3.80	61.32	-	-	1.56
in505	1500	0.08	84860.165	2.16	2.82	62.09	-	-	1.56
in506	1500	0.08	84623.414	1.41	2.32	65.92	-	-	1.34
in507	1500	0.08	90288.472	2.82	3.73	68.99	-	-	1.32
in508	1500	0.07	86853.500	1.24	1.60	58.26	-	-	1.11
in509	1500	0.08	88316.087	3.59	4.37	64.39	-	-	1.80
in510	1500	0.08	89014.137	1.29	2.19	64.39	-	-	1.15
in601	1500	0.09	108800.445	4.52	5.51	76.04	-	-	2.39
in602	1500	0.09	105611.476	2.01	2.98	70.61	-	-	1.75
in603	1500	0.09	105121.021	1.84	2.56	65.96	-	-	1.35
in604	1500	0.10	107733.805	3.86	5.71	84.58	-	-	3.09
in605	1500	0.10	109840.984	3.72	5.75	75.27	-	-	2.22
in606	1500	0.10	107113.067	2.72	4.37	77.60	-	-	2.13
in607	1500	0.10	113180.284	4.20	6.65	78.37	-	-	2.38
in608	1500	0.10	105266.107	2.21	3.72	77.60	-	-	2.17
in609	1500	0.10	109472.332	2.77	3.78	69.84	-	-	1.86
in610	1500	0.11	113716.965	6.14	10.03	94.67	-	-	3.70

In the future, we plan to study the impact of vertex ordering and unit clause ordering in MW-CLQ. It is also interesting to use MaxSAT reasoning to solve other combinatorial optimization problems, especially the weighted version, with the LW encoding.

6. Acknowledgment

This research was partly supported by NSFC (Grant No. 61421003), the fund of the State Key Lab of Software Development Environment (Grant No. SKLSDE-2015ZX-05) and the Chinese State Key Laboratory of Software Development Environment Open Fund (Grant No. SKLSDE-2012KF-07).

References

- Ansótegui, C., Bonet, M. L., & Levy, J. (2009). Solving (weighted) partial maxsat through satisfiability testing. In *Theory and Applications of Satisfiability Testing-SAT 2009*, pp. 427–440.
- Benlic, U., & Hao, J. K. (2013). Breakout local search for maximum clique problems. *Computers & Operations Research*, 40, 192–206.
- Cai, S., Su, K., Luo, C., & Sattar, A. (2013). NuMVC: An efficient local search algorithm for minimum vertex cover. *J. Artif. Intell. Res.(JAIR)*, 46, 687–716.
- Cai, S., Su, K., & Sattar, A. (2011). Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9), 1672–1696.
- Chen, J. (2010). A new SAT encoding of the at-most-one constraint. *Proc. Constraint Modelling and Reformulation*.
- Davies, J., & Bacchus, F. (2013). Postponing optimization to speed up MAXSAT solving. In *Principles and practice of constraint programming*, pp. 247–262. Springer.
- Downey, R. G., & Fellows, M. R. (1995). Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4), 873–921.
- Fahle, T. (2002). Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In *Proc. of ESA-2002*, pp. 485–498.
- Fang, Z., Chu, Y., Qiao, K., Feng, X., & Xu, K. (2014a). Combining edge weight and vertex weight for minimum vertex cover problem. In *Frontiers in Algorithmics*, pp. 71–81. Springer.
- Fang, Z., Li, C. M., Qiao, K., Feng, X., & Xu, K. (2014b). Solving maximum weight clique using maximum satisfiability reasoning. In *The 21st European Conference in Artificial Intelligence*.
- Feige, U. (2004). Approximating maximum clique by removing subgraphs. *SIAM Journal on Discrete Mathematics*, 18(2), 219–225.
- Freeman, J. W. (1995). *Improvements to propositional satisfiability search algorithms*. Ph.D. thesis, Citeseer.
- Guo, Y., Lim, A., Rodrigues, B., & Zhu, Y. (2006). Heuristics for a bidding problem. *Computers & operations research*, 33(8), 2179–2188.
- Ignatiev, A., Morgado, A., & Marques-Silva, J. (2014). On reducing maximum independent set to minimum satisfiability. In *Theory and Applications of Satisfiability Testing–SAT 2014*, pp. 103–120. Springer.

- Karp, R. M. (1972). Reducibility among combinatorial problems.. Springer.
- Kibanov, M., Atzmueller, M., Scholz, C., & Stumme, G. (2014). Temporal evolution of contacts and communities in networks of face-to-face human interactions. *Science China Information Sciences*, 57(3), 1–17.
- Konc, J., & Janezic, D. (2007). An improved branch and bound algorithm for the maximum clique problem. *Communications in Mathematical and in Computer Chemistry*, 58, 569–590.
- Kügel, A. (2010). Improved exact solver for the weighted Max-SAT problem. In *Workshop Pragmatics of SAT*, Vol. 436.
- Kügel, A. (2012). Natural Max-SAT encoding of Min-SAT. In *Learning and Intelligent Optimization*, pp. 431–436. Springer.
- Kumlander, D. (2004). A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. In *Proc. of MOC'04.*, pp. 202–208.
- Kumlander, D. (2008). On importance of a special sorting in the maximum-weight clique algorithm based on colour classes. In *Modelling, computation and optimization in information systems and management sciences*, pp. 165–174. Springer.
- Larrosa, J., Heras, F., & de Givry, S. (2008). A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172, 204–233.
- Lau, H. C., & Goh, Y. G. (2002). An intelligent brokering system to support multi-agent web-based 4 th-party logistics. In *Tools with Artificial Intelligence, 2002.(ICTAI 2002). Proceedings. 14th IEEE International Conference on*, pp. 154–161. IEEE.
- Li, C. M., & Anbulagan, A. (1997). Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th international joint conference on Artificial intelligence-Volume 1*, pp. 366–371. Morgan Kaufmann Publishers Inc.
- Li, C. M., Fang, Z., & Xu, K. (2013). Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pp. 939–946. IEEE.
- Li, C. M., Manyà, F., & Planes, J. (2005). Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers lower bounds in branch and bound max-sat solvers. In *Proc. of CP'05*, Vol. 3709, pp. 403–414. LNCS.
- Li, C. M., Manyà, F., & Planes, J. (2007). New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30, 321–359.
- Li, C. M., Manyà, F., & Planes, J. (2006). Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *AAAI*, Vol. 6, pp. 86–91.
- Li, C. M., & Quan, Z. (2010a). Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, Vol. 1, pp. 344–351. IEEE.
- Li, C. M., & Quan, Z. (2010b). An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In *Proc. of AAAI-2010*, pp. 128–133.
- Li, C. M., Zhu, Z., Manyà, F., & Simon, L. (2012). Optimizing with minimum satisfiability. *Artificial Intelligence*, 190, 32–44.

- Liu, T., Lin, X., Wang, C., Su, K., & Xu, K. (2011). Large hinge width on sparse random hyper-graphs. In *IJCAI*, Vol. 2011, pp. 611–616. Citeseer.
- Ma, T., & Latecki, L. J. (2012). Maximum weight cliques with mutex constraints for video object segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 670–677. IEEE.
- Mascia, F., Cilia, E., Brunato, M., & Passerini, A. (2010). Predicting structural and functional sites in proteins by searching for maximum-weight cliques. In *Proc. of AAAI-2010*, pp. 1274–1279. AAAI.
- Östergård, P. (2001). A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing*, 8, 424–436.
- Östergård, P. (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120, 197–207.
- Pullan, W. (2008). Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 19, 117–134.
- Pullan, W., & Hoos, H. H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25, 159–185.
- Régin, J. C. (2003). Solving the maximum clique problem with constraint programming. In *Proc. of CPAIOR-2003*, pp. 634–648.
- Sghir, I., Hao, J.-K., Jaafar, I. B., & Ghédira, K. (2014). A recombination-based tabu search algorithm for the winner determination problem. In *Artificial Evolution*, pp. 157–167. Springer.
- Shimizu, S., Yamaguchi, K., Saitoh, T., & Masuda, S. (2012). Some improvements on Kumlander’s maximum weight clique extraction algorithm. In *Proc. of the International Conference on Electrical, Computer, Electronics and Communication Engineering*, pp. 307–311.
- Shimizu, S., Yamaguchi, K., Saitoh, T., & Masuda, S. (2013). Optimal table method for finding the maximum weight clique. In *WSEAS International Conference. Proceedings. Recent Advances in Computer Engineering Series*, No. 12. WSEAS.
- Tomita, E., & Kameda, T. (2007). An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 37, 95–111.
- Tomita, E., & Seki, T. (2003). An efficient branch-and-bound algorithm for finding a maximum clique. In *Proc. Discrete Mathematics and Theoretical Computer Science*, Vol. 2731, pp. 278–289.
- Wu, Q., Hao, J. K., & Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196, 611–634.
- Wu, Q., & Hao, J.-K. (2015). Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications*, 42(1), 355–365.
- Xu, K., Boussemart, F., Hemery, F., & Lecoutre, C. (2007). Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171, 514–534.
- Xu, K., & Li, W. (2000). Exact phase transitions in random constraint satisfaction problems. *J. Artif. Intell. Res. (JAIR)*, 12, 93–103.

- Xu, K., & Li, W. (2006). Many hard examples in exact phase transitions. *Theoretical Computer Science*, 355(3), 291–302.
- Yamaguchi, K., & Masuda, S. (2008). A new exact algorithm for the maximum weight clique problem. In *Proc. of ITC-CSCC'08*, pp. 317–320.
- Zhang, D., Javed, O., & Shah, M. (2014a). Video object co-segmentation by regulated maximum weight cliques. In *Computer Vision—ECCV 2014*, pp. 551–566. Springer.
- Zhang, W., Nie, L., Jiang, H., Chen, Z., & Liu, J. (2014b). Developer social networks in software engineering: construction, analysis, and applications. *Science China Information Sciences*, 57(12), 1–23.
- Zhian, H., Sabaei, M., Javan, N. T., & Tavallaie, O. (2013). Increasing coding opportunities using maximum-weight clique. In *Computer Science and Electronic Engineering Conference (CEEC), 2013 5th*, pp. 168–173. IEEE.
- Zhu, Z., Li, C. M., Manyà, F., & Argelich, J. (2012). A new encoding from MinSAT into MaxSAT. In *Principles and Practice of Constraint Programming*, pp. 455–463. Springer.
- Zuckerman, D. (2006). Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pp. 681–690. ACM.