ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-303

Solving Maximum Weight Clique Using Maximum Satisfiability Reasoning

Zhiwen Fang[1,2] and **Chu-Min Li**[2] and **Kan Qiao**[3] and **Xu Feng**[1] and **Ke Xu**[1]

Abstract. Satisfiability (SAT) and maximum satisfiability (MaxSAT) techniques are proved to be powerful in solving combinatorial optimization problems. In this paper, we encode the maximum weight clique (MWC) problem into weighted partial MaxSAT and use MaxSAT techniques to solve it. Concretely, we propose a new algorithm based on MaxSAT reasoning called *Top-k* failed literal detection to improve the upper bound for MWC, and implement an exact branch-and-bound solver for the MWC problem called MaxWClq based on the *Top-k* failed literal detection algorithm. To our best knowledge, this is the first time that MaxSat techniques are integrated to solve the MWC problem. Experimental evaluations on the broadly used DIMACS benchmark, BHOSLIB benchmark and random graphs show that MaxWClq outperforms state-of-the-art exact algorithms on the vast majority of instances. In particular, our algorithm is surprisingly powerful for dense and hard graphs.

1 Introduction

Given an undirected graph $G = (V,E)$, a clique is a complete subgraph of G where each pair of vertices is connected. The maximum clique (MC) problem (MCP) calls for finding the clique with the largest cardinality, denoted by $\omega(G)$. MCP is a prominent combinatorial optimization problem, which appears in a variety of real-word applications like social network analysis and bioinformatics.

A vertex-weighted undirected graph is an undirected graph $G = (V,E)$, in which every vertex is associated with a nonnegative value as the weight. The weight of a clique is defined as the total weight of vertices in it. Given a vertex-weighted undirected graph, the maximum weight clique (MWC) problem (MWCP) consists in finding a clique with the largest weight. MWCP is an important generalization of MCP and has wide applications in real-world problems such as protein structure predictions and combinatorial auctions. Much more effort has been devoted to MCP than to MWCP.

MCP is NP-hard and its decision problem is one of Karp's 21 NP-complete problems. A huge amount of effort has been devoted to solve MCP and MWCP. Generally speaking, there are two types of algorithms for MCP. One is approximation algorithms including the stochastic local search, e.g., [17, 4, 2, 3]. The other is exact algorithms including the branch-and-bound search, e.g., [15, 16, 19, 8, 13, 23, 14]. Approximation 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 $P = NP$.

Exact branch-and-bound algorithms often need a high-quality bound obtained within reasonable time for pruning. How to obtain such an upper bound is a challenging problem in solving MCP and MWCP. Independent set partition (Graph coloring) and MaxSAT reasoning are the two most powerful techniques to solve MCP, which have been used in most state-of-the-art solvers. For example, Fahle [5] uses the constructive heuristic DSATUR to color vertices one by one according to their degrees. The number of colors is used as the upper bound. MCQ [19] uses a heuristic vertex order for independent set partition. MaxCliqueDyn [6] improves MCQ by computing the degree of vertices dynamically. MaxCLQ [13] encodes MaxClique problem into MaxSAT based on the independent set partition of the graph, then it applies MaxSAT reasoning to improve the upper bound. The performance of MaxCLQ shows that MaxSAT reasoning is a powerful technique for MCP. IncMaxClq [14] combines an incremental upper bound and MaxSAT, which allows one to obtain an even tighter bound with less time consumption. Besides independent set partition and MaxSAT techniques, other approaches are also used to solve MCP. For instance, Régin [18] computes an upper bound using a matching algorithm.

A straightforward way to solve MWCP is to extend techniques for MCP. For instance, Cliquer [16, 15] is one of state-of-the-art exact solvers for both MCP and MWCP, and it deals with MCP and MWCP using similar methods. It sets a static ordering of vertices based on an independent set partition and tries to find a larger clique by adding new vertices on by one in this ordering. Meanwhile, a value is associated to each vertex as an upper bound for the maximum clique containing it. Kumlander [8] adopts the independent set partition method and uses the sum of the maximum weight of vertex in each independent set as an upper bound for MWC. Also, some specific techniques are introduced to MWCP. For example, Yamaguchi and Masuda [23] 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.

Observe that the independent set partition has been used for solving both MCP and MWCP. However, MaxSAT reasoning, which proves to be very effective for MCP, has not been extended for MWCP to our best knowledge. Indeed, while many approaches for unweighted combinatorial optimization problems such as MaxSAT can be efficiently extended to the weighted case without substantial difficulty, extending MaxSAT reasoning from MCP to MWCP is not easy to be efficient because of the vertex weights. More precisely, when a solving technique for unweighted MaxSAT is extended to weighted MaxSAT in the literature, it is usually assumed all variables in a weighted clause have the same importance, although different

[1] State Key Lab. of Software Development Environment, Beihang University, Beijing, China, email: zhiwenf@gmail.com; isaiah.feng@gmail.com; kexu@nlsde.buaa.edu.cn
[2] MIS, Université de Picardie Jules Verne, Amiens, France, email: chumin.li@u-picardie.fr
[3] Department of Computer Science, Illinois Institute of Technology, Chicago IL, USA, email: kqiao@iit.edu

clauses can have different weights in a weighted MaxSAT instance. However, when the MaxSAT encoding of MCP is extend to MWCP directly, the variables in a clause DO NOT have the same importance, and clauses should be dynamically split taking into account the different importance of their variables.

In this paper, we encode MWCP into weighted partial MaxSAT, in which not only clauses have different weights, but variables in each clause also have different weights. Then we propose an algorithm named *Top-k* failed literal detection to handle this specific weighted MaxSAT instance for computing tight upper bounds for MWC. The idea of *Top-k* failed literal detection is to split the weight of a variable when necessary to generate more clauses, allowing to detect more conflicts. Consequently, tighter upper bounds can be obtained based on MaxSAT reasoning for MWC.

The rest of the paper is organized as follows. In the next section, we introduce some necessary background knowledge before describing a basic branch-and-bound algorithm for MWC. The next two sections present the method to encode from MWCP into MaxSAT and the algorithm to compute a tight upper bound based on MaxSAT reasoning respectively. Experimental results are shown in Section 6. Finally, we give some concluding remarks.

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

Input: MaxSAT formula F and a stack of unit clauses S
Output: return an inconsistent subset of soft clauses , $\emptyset$ if no inconsistent subset is found

```
1  begin
2      while S is not empty do
3          pop a unit clause uc from S
4          l ← literal in uc
5          foreach clause cl ∈ F contains l do
6              satisfy cl
7          foreach clause cl ∈ F constains l̄ do
8              remove l̄ from cl
9              if cl is empty then
10                 return the inconsistent subset of soft clauses
11             if cl is a unit clause then
12                 push cl into S
13     return ∅
```

2 Preliminaries

An undirected graph $G = (V,E)$ comprises a set $V = \{v_1, v_2,..., v_n\}$ of n vertices together with a set E of m edges. The density of G is computed as $2m/(n(n-1))$. A clique of G is a subset $C \subseteq V$ such that every pair of vertices in C is connected by an edge. C is a maximum clique if no clique with larger size exists in G. Let $V' \subseteq V$, the subgraph of G induced by V' is $G(V') = (V',E')$, where $E' = \{\{v_i,v_j\} | \ v_i,v_j \in V' \wedge \{v_i,v_j\} \in E\}$. For each vertex v, $\Gamma(v) = \{u | \ \{u,v\} \in E\}$ is the neighbor vertices 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 $G \backslash v$ for the one induced by $V \backslash \{v\}$. Note that, for any given vertex v, the maximum clique of graph G exists either in G_v or in $G \backslash v$. An independent set of G is an subset $I \subseteq V$ such that any pair of vertices in I are disconnected. If G can be partitioned into k independent sets, then $\omega(G) \leq k$, since each independent set can contribute at most one vertex to the clique.

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 each vertex v is associated with a nonnegative value $w(v)$ as its weight. Given a weighted graph $G = (V, E, w)$, the weight of a clique C in G is the total weight of vertices in C, denoted by $W(C)$. The maximum weight clique problem consists of finding a clique with the largest weight, denoted by $MW(G)$. MWCP is an important generalization of MCP and it is also a NP-hard problem.

In propositional logic, a variable x may take value 0 (false) or 1 (true). A literal l is a variable x or its negation $\overline{x}$. A clause $c = l_0 \vee l_1 \vee ... \vee l_k$ is a disjunction of literals and can be expressed in a set of literals: $\{l_1, l_2, ..., l_k\}$. 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 and an empty clause is the clause without any literal. A conjunctive normal form (CNF) formula $F = c_1 \wedge c_2 \wedge ... \wedge c_m$ is a conjunction of clauses. Given a formula F on the set of variables $\{x_1, x_2, ..., x_n\}$, the maximum satisfiability problem is to find an assignment satisfying the maximum number of clauses. The MaxSAT problem is partial when it contains hard clauses, i.e., clauses that must be satisfied in all solutions, and soft clauses which can be unsatisfied. The partial MaxSAT problem is to find an assignment to maximize the number of soft clauses satisfied while satisfying all hard clauses. Notice that the MaxSAT problem is a particular partial MaxSAT problem containing only soft clauses. A weighted clause is a pair (c,w), where c is the clause and w is a nonnegative value as its weight. The weighted MaxSAT problem is to find an assignment to maximize the total weight of satisfied clauses. A weighted partial MaxSAT problem calls for finding an assignment maximizing the total weight of the satisfied soft clauses while satisfying all the hard clauses.

To maximize the number of satisfied clauses equals to minimize the number of unsatisfied clauses, therefore many branch-and-bound based algorithms for MaxSAT underestimate the minimum number of unsatisfied clauses as a bound [12, 9]. Detecting disjoint inconsistent subsets is proved to be very powerful in computing such a bound. A subset of soft clauses is called $inconsistent$ if this subset together with hard clauses results in a contradiction (an empty clause). The number of disjoint inconsistent subsets is a lower bound of the number of unsatisfied clauses, since there is at least one unsatisfied clause in each inconsistent subset. Unit propagation (UP) is an effective technique widely used in SAT and MaxSAT solvers [10, 11]. The pseudo-code allowing to find an inconsistent subset of soft clauses based on UP is given in Algorithm 1. The algorithm works as follows, it uses a stack S to store all unit clauses in F and does UP 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 as possible. Notice that soft clauses involved in an inconsistent subset should not be used to produce other inconsistent subsets.

Failed literal detection is proposed to enhance unit propagation. A literal l is called *failed literal* in the formula F, if unit propagation in $F \cup \{l\}$ produces an empty clause. Let *InconSet*(l) be the set of soft clauses used to produce the empty clause. If both l and $\overline{l}$ are failed literals, the union of the clauses used to produce the two empty clauses, i.e., *InconSet*(l) $\cup$ *InconSet*($\overline{l}$), constitutes an inconsistent subset. For the MaxSAT instance encoding a MCP instance, one does not detect whether or not a negative literal is failed, because a variable only has positive occurrence in a soft clause. Therefore, given a soft clause $c = l_1 \vee l_2 \vee ... \vee l_t$, if l_1, l_2, ..., l_t are all failed literals, then $\{c\} \cup$ *InconSet*(l_1) $\cup$ *InconSet*(l_2) $\cup$ ... $\cup$ *InconSet*(l_t) is an inconsistent subset. Refer to [13] for more details.

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

Input: A vertex-weighted graph G=(V, E, W), a clique C, and a lower bound LB
Output: A maximum weight clique with weight larger than LB, $\emptyset$ if no such clique is found

```
1  begin
2      if |V| = 0 then
3          return C
4      UB ← overestimate(G)+W(C)
5      if UB ≤ LB then
6          return ∅
7      v ← select(V)
8      C1 ← MaxWClq(Gv \ v, C ∪ v, LB)
9      C2 ← MaxWClq(G \ v, C, max(LB,W(C1)))
10     return the one with larger weight in {C1, C2}
```

3 MaxWClq: A New Algorithm for MWCP

We propose a basic branch-and-bound algorithm, called MaxWClq, for MWCP in this section. Algorithm 2 shows the pseudo-code of MaxWClq.

Given any v in graph G, MWC is either a clique in G_v containing v or one in $G\backslash v$ without v. Thus MaxWClq tries to find the optimal solution in G_v and $G\backslash v$ respectively. To make MaxWClq fast, a high-quality upper bound is needed to prune useless search that cannot give cliques with weight larger than LB, which is always the weight of the best solution found so far. At the same time, the ordering of the vertices to determine which v to branch on next, also plays an important role in search. In this paper, we use a very simple vertex ordering to select the vertex with the largest weight, breaking ties in favor of the one with higher degree. In fact, a better heuristic vertex ordering is one direction of our future work. Our work focus on how to compute a tight upper bound within reasonable time, which is presented in the following sections.

4 Encode Maximum Weight Clique into MaxSAT

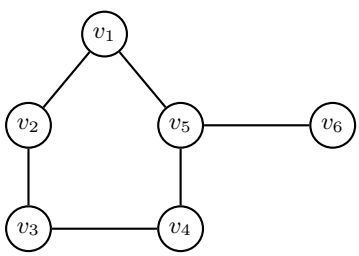

Figure 1. Graph with w_1=1, w_2=7, v_3=2, w_4=3, w_5=4, w_6=6

Given an undirected graph $G = (V, E)$, we introduce a boolean variable x_i for each vertex v_i to indicate whether or not v_i is in the clique, i.e., $x_i = 1$ if and only if v_i is contained in the clique. For the weighted graph $G = (V, E, w)$, each variable x_i can be associated with the value $w_i = w(v_i)$ as its weight.

To encode MWCP into weighted partial MaxSAT, the set of hard clauses is always the same for different encodings. For each unconnected pair of vertices v_i and v_j ($\{v_i, v_j\} \notin E$), $\overline{x}_i \vee \overline{x}_j$ should be added into the set of hard clauses to say that they cannot be contained in the same clique. For the example in Figure 1, the set of hard clauses is $\{\overline{x}_1 \vee \overline{x}_3, \overline{x}_1 \vee \overline{x}_4, \overline{x}_1 \vee \overline{x}_6, \overline{x}_2 \vee \overline{x}_4, \overline{x}_2 \vee \overline{x}_5, \overline{x}_2 \vee \overline{x}_6, \overline{x}_3 \vee \overline{x}_5, \overline{x}_3 \vee \overline{x}_6, \overline{x}_4 \vee \overline{x}_6\}$.

Given a weighted undirected graph G = (V,E,w), the direct way to encode MWCP into weighted MaxSAT is to associate each vertex v_i with a weighted soft clause (x_i,w_i). Maximizing the weight of satisfied soft clauses while satisfying all the hard clauses gives rise to a maximum weight clique. Take Figure 1 for example, the set of soft clauses is {(x_1,1), (x_2,7), (x_3,2), (x_4,3), (x_5,4), (x_6,6)}. One drawback of this encoding is that the soft clauses cannot capture the structure of the graph well, since it generates soft clauses regardless of the connections between vertices.

MaxCLQ [13] introduces an approach based on independent set partition to encode MCP into MaxSAT. Suppose that a graph can be partitioned into k independent sets. For each independent set, a soft clause is created to be the $disjunction$ of the variables corresponding the vertices in the independent set. We extend this encoding to weighted graph for MWCP as follows.

Definition 1 *Given a weighted graph G = (V,E,w), let P = {I_1, I_2, ..., I_k} be an independent set partition of G. The independent set based MaxSAT encoding of MWCP is defined as follows: (1) each vertex v_i is represented by a weighted boolean variable (x_i,w_i) where w_i = $w(v_i)$. (2) for independent set I_i = $\{v_{i_1}, v_{i_2}, ..., v_{i_j}\}$, a weighted soft clause (c_i,$w(c_i)$) is added, where $c_i = x_{i_1} \vee x_{i_2} \vee ... \vee x_{i_j}$ and $w(c_i) = \max_{x_j \in c_i} w_j$. (3) a hard clause $\overline{x}_i \vee \overline{x}_j$ is added for each pair of vertices $\{v_i, v_j\} \notin E$.*

A weighted clause can also be presented as a set. For instance, in Definition 1 (c_i,$w(c_i)$) can be also represented as c_i = {(x_{i_1},w_{i_1})), (x_{i_2},w_{i_2})), ..., (x_{i_j},w_{i_j}))}. c_i is *ordered* if $w_{i_1} \geq w_{i_2} \geq ... \geq w_{i_j}$.

Example 1 Refer to Figure 1, one possible independent set partition is {{v_1, v_4, v_6}, {v_2, v_5}, {v_3}}. By Definition 1, soft clauses in the encoding based on the partition are ({(x_6,6), (x_4,3), (x_1,1)}, 6), ({(x_2,7), (x_5,4)}, 7) and ({(x_3,2)}, 2), respectively.

A trivial upper bound for MWC is to sum the vertex weight to obtain the upper bound based on the direct encoding. Refer to Figure 1, this upper bound is 23. A tighter bound can be obtained using the MaxSAT formula based on the independent set partition.

Proposition 1 *Given a weighted undirected graph G=(V, E, w), Let P={I_1, I_2, ..., I_k} be an independent set partition of G and {c_1, c_2, . . ., c_k} be the set of soft clauses in the MaxSAT encoding based on P, then MW(G) $\leq \sum_{1 \leq i \leq k} w(c_i)$.*

The correctness of Proposition 1 is straightforward since every independent set can contribute at most one vertex with the largest weight to the clique. Take Figure 1 for example, the MaxSAT formula in Example 1 improves the upper bound to 15.

It is not hard to verify that the maximum weighted clique of Figure 1 is {v_5,v_6} with weight 10. In fact, the minimal number of independent set, i.e., the chromatic number, in Figure 1 is 3, since it is an imperfect graph and the chromatic number can be larger than the size of maximum clique. In other words, the best upper bound may not be obtained based on the independent set partition of an imperfect graph. How to achieve a tighter upper bound with MaxSAT techniques for a weighted graph will be presented in the next section.

5 Upper Bound for MWC

In partial MaxSAT, a subset of soft clauses is called inconsistent if the subset together with hard clauses can result in a contradiction.

Branch-and-bound MaxSAT solvers often detect disjoint inconsistent subsets to obtain a tight upper bound. The weight of an inconsistent subset of soft clauses S is computed as follows.

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

Proposition 2 *Given a weighted undirected graph G=(V, E, w), let F be the set of soft clauses in the independent set based MaxSAT encoding, if there are s disjoint inconsistent subsets: S_1, S_1, ..., S_s detected in F, then $\sum_{c \in F} w(c)$-$\sum_{1 \le i \le s} W(S_i)$ is an upper bound for the optimal solution of F.*

UP detects inconsistent subset of soft clauses as follows.

Example 2 In the independent based MaxSAT encoding of the graph in Figure 1 presented in Example 1, the set of soft clauses is $\{(x_1 \vee x_4 \vee x_6, 6), (x_2 \vee x_5, 7), (x_3, 2)\}$ while the set hard clauses is$\{\overline{x}_1 \vee \overline{x}_3, \overline{x}_1 \vee \overline{x}_4, \overline{x}_1 \vee \overline{x}_6, \overline{x}_2 \vee \overline{x}_4, \overline{x}_2 \vee \overline{x}_5, \overline{x}_2 \vee \overline{x}_6, \overline{x}_3 \vee \overline{x}_5, \overline{x}_1 \vee \overline{x}_6, \overline{x}_4 \vee \overline{x}_6\}$. The initial upper bound based on Proposition 1 is 15. Let us see how UP allows to improve the upper bound. Set x_3 = 1 to satisfy unit clause $(x_3, 2)$, then $\overline{x}_3$ will be removed from clauses $\overline{x}_1 \vee \overline{x}_3$, $\overline{x}_3 \vee \overline{x}_5$, $\overline{x}_3 \vee \overline{x}_6$. Three new unit clauses: $\overline{x}_1$, $\overline{x}_5$, $\overline{x}_6$ imply that x_1 = 0, x_5 = 0, x_6 = 0. After removing x_1 and x_6 from $x_1 \vee x_4 \vee x_6$ and x_5 from $x_2 \vee x_5$, two new unit clauses: x_4 and x_2 make the hard clause $\overline{x}_2 \vee \overline{x}_4$ empty. So far, an inconsistent soft clause subset containing all three soft clauses is detected and the weight of the subset is 2. Using Proposition 2, it 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 can generate more soft clauses. To generate more soft clauses, a simplification rule is needed, which transforms a formula F into an equivalent, but presumably easier formula F'. In this section, we propose two simplification rules for MaxSAT clauses with different variable weights.

Rule 1 Given a weighted undirected graph $G = (V,E,w)$ and an inconsistent subset S of soft clauses in the independent set based MaxSAT encoding of G, let $\delta = W(S)$, then for every ordered soft clause $c = \{(x_1,w_1), (x_2,w_2), \ldots, (x_t,w_t)\}$ in S, c can be replaced by $c' = \{(x_1,\min(w_1,\delta)), (x_2,\min(w_2,\delta)), ..., (x_t,\min(w_t,\delta))\}$ and $c'' = \{(x_1,\max(w_1\text{-}\delta,0)), (x_2,\max(w_2\text{-}\delta,0)), ..., (x_t,\max(w_t\text{-}\delta,0))\}$.

Rule 1 is used to simplify all soft clauses in an inconsistent subset. It does not change the total weight of soft clauses. Clause c' contains the same collection of literals as c with $w(c') = \delta$. Recall that the variable with weight 0 in c'' should be removed from it.

Proposition 3 *Let $S = \{c_1, c_2, ..., c_i\}$ be an inconsistent subset of soft clauses with weight $\delta = w(S)$, apply Rule 1 for S to replace every c by c' and c'', then $S' = \{c'_1, c'_2, ..., c'_i\}$ is still an inconsistent soft clause subset with weight w.*

Recall that each soft clause $c' \in S'$ involves the same collection of literals as the corresponding soft clause $c \in S$ and every clause $c' \in S$ has the same weight equaling δ, thus S' is an inconsistent soft clause with the weight δ, therefore $S'' = \{c''_1, c''_2, \ldots, c''_i\}$ can still be used to improve the upper bound.

Example 3 As presented in Example 2, $\{\{(x_6,6), (x_4,3), (x_1,1)\}, \{(x_2,7),(x_5,4)\}, \{(x_3,2)\}\}$ is an inconsistent subset with weight 2. Applying Rule 1, it is replaced by $\{\{(x_4,2), (x_6,2), (x_1,1)\}, \{(x_6,4), (x_4,1)\}, \{(x_2,2),(x_5,2)\}, \{(x_2,5),(x_5,2)\}, \{(x_3,2)\}\}$. Notice that $\{(x_1 \vee x_4 \vee x_6,2), (x_2 \vee x_5,2), (x_3,2)\}$ is still an inconsistent subset of soft clause with the weight 2. The remaining clauses $\{\{(x_6,4), (x_4,1)\}, \{(x_2,5),(x_5,2)\}\}$ can be used to improve the upper bound.

Algorithm 3: $overestimate(G)$, overestimate an upper bound for MWC by *Top-k* failed literal detection.

```
Input: A weighted graph G=(V, E, w)
Output: An upper bound for maximum weight clique in G
1  begin
2      partition G into k independent sets:I_1, I_2,...,I_k
3      encode G into a weighted partial MaxSAT formula F
4      UB ← Σ_{c∈F} w(c)
5      mark all soft clauses available
6      while existing soft clause c∈F is available do
7          sort literals of c in non-increasing weight order
8          k ← 0, S ← ∅
9          while k < length(c) do
10             if l_{k+1} ∈ c is not a failed literal then
11                 break
12             S ← S ∪ InconSet(l_{k+1})
13             k ← k + 1
14         if k > 0 then
15             if k = length(c) then
16                 S ← S∪{c}
17             else if k = length(c) then
18                 apply Rule 2 to replace c by c' and c''
19                 S ← S∪{c'}
20             UB ← UB−W(S)
21             apply Rule 1 to simplify S
22             mark all soft clauses contained in the inconsistent
               subset unavailable
23     return UB
```

After the simplification, unit propagation cannot be used since there is no unit clause and another approach named failed literal detection does not work either because no clause contains only failed literals [13]. We propose *Top-k* failed literal detection algorithm to detect more conflicts. An ordered soft clause $c = \{(x_1,w_1), (x_2,w_2), ..., (x_t,w_t)\}$ is *Top-k* failed if $x_1,x_2,...,x_k$ are all failed literals.

Rule 2 Given an ordered clause $c = \{(x_1,w_1), (x_2,w_2), \ldots, (x_l,w_l)\}$ and an integer $k < l$, let $\delta = w_1\text{-}w_{k+1}$, then c can be replaced by $c' = \{(x_1,\delta),(x_2, min(w_2,\delta), \ldots, (x_k,min(w_k,\delta)\}$ and $c'' = \{(x_1,max(w_1\text{-}\delta,0), (x_2,max(w_2\text{-}\delta,0)), \ldots, (x_k,max(w_k\text{-}\delta,0)), (x_k,w_{k+1}), \ldots, (x_t,w_t)\}$.

Rule 2 is used to simplify a *Top-k* failed soft clause. We can verify that $w(c) = w(c')+w(c'')$, therefore Rule 2 does not change the total weight of soft clauses neither. Note that all literals in c' are failed literals, therefore $\{c'\}$ together with *InconSet*$(x_1) \cup$ *InconSet*$(x_2) \cup \ldots \cup$ *InconSet*(x_k) is an inconsistent soft clause subset.

Example 4 Continuing with Example 3, we can easily verify $\{(x_2,5),(x_5,2)\}$ is *Top-k* failed with k = 1, since setting x_2=1 results in x_4=0 and x_6=0 so that $\{(x_4,1), (x_6,4)\}$ becomes an empty clause. With Rule 2, $\{(x_2,5),(x_5,2)\}$ is replaced by $\{(x_2,3)\}$ and $\{(x_2,2),(x_5,2)\}$. Then $\{(x_2,3)\}$ together with $\{(x_6,4), (x_4,1)\}$ constitutes an inconsistent subset with the weight 3. Apply Rule 1 to replace it by $\{\{(x_2,3)\}, \{(x_6,3), (x_4,1)\}, \{(x_6,1)\}\}$. At last, we detect that $\{\{(x_2,3)\}, \{(x_6,3), (x_4,1)\}\}$ is an inconsistent subset with the weight 3. Therefore, the upper bound can be improved by 3 and *Top-k* failed literal detection achieves the tightest upper bound 10.

The algorithm based on *Top-k* failed literal detection is presented in Algorithm 3. It works as follows. First, the weighted graph is encoded into weighted partial MaxSAT based on an independent set

partition. Vertices are sorted by the decreasing order of their weights and are inserted into an independent set one by one. Suppose the current independent sets are I_1, I_2, ..., I_k, we try to find an independent set from I_1 to I_k to insert the vertex, if no such an independent set exists, a new independent set I_{k+1} is created. We use the total weights of soft clauses as an initial upper bound and then use *Top-k* failed literal detection algorithm to detect as many disjoint inconsistent subsets as possible. Add c into S if an inconsistent subset is detected or apply Rule 2 to simplify c and add c' into S if c is *Top-k* failed. Afterwards, use Rule 1 to simplify S. Note that an inconsistent subset is detected if c is *Top-k* failed while $k = length(c)$. At last, the initial upper bound can be improved by the total weight of the disjoint inconsistent subsets detected.

6 Empirical Evaluation

Table 1. Average runtime in sec on 50 graphs each case. $|V|$ stands for vertex number, D for density, $\overline{W}$ is average weight of MWC. '-' means none of 50 graphs cat be solved within 3600s.

$\|V\|$	D	$\overline{W}$	Cliquer	MaxWClq
150	0.90	3392	22.97	**0.76**
150	0.95	4735	959.1	**0.88**
200	0.80	3275	4.21	**1.15**
200	0.90	5055	1409	**22.09**
200	0.95	7269	-	**70.75**
300	0.70	2465	2.03	**1.32**
300	0.80	3341	81.38	**19.27**
300	0.90	5342	-	**1205**
500	0.60	2283	**4.27**	6.91
500	0.70	2955	139.7	**117.3**
600	0.60	2509	**25.26**	39.70
600	0.70	3283	1395	**1039**

We evaluate the performance of MaxWClq on the widely used DIMACS benchmark[4] and BHOSLIB benchmark[5] as well as random graphs. We convert a non-weighted graph into a weighted graph as follows. For each vertex v_i, we set $w_i = i \bmod 200 + 1$ as the weight of v_i. This method was initially proposed in [17] and has been used as a standard converting approach to generate weighted graphs from non-weighted benchmarks [20, 2].

The DIMACS benchmark is taken from the Second DIMACS Implementation Challenge, which has been used widely for benchmarks purposes in the literature on MCP algorithms. The DIMACS instances are generated from problems in coding theory, fault diagnosis problems and so on. BHOSLIB benchmark is based on a CSP model named RB [22, 21]. It arises from the SAT competition 2004 and is also widely used to evaluate maximum clique and minimum vertex cover solvers. Weighted graphs converted from both benchmarks are also used to evaluate approximation algorithms for MWCP in [17, 20, 2].

Random graphs allow one to show asymptotic behavior of a solver. In our experiments, random graphs are generated based on the $G(n, p)$ model, in which each edge is included in the graph with probability p independently from any other edge. Then they are converted in a weighted graph with the standard approach.

[4] http://cs.hbg.psu.edu/txn131/clique.html

[5] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

We compare our solver with other state-of-the-art exact solvers for MWC. Although there are many exact solvers for MC, to our best knowledge, only Cliquer [16, 15] can handle weighted instances. We compare MaxWClq with the latest version of Cliquer [6] updated in 2012. Algorithms presented in [8, 23] are not available to us. We also compare MaxWClq with some state-of-the-art MaxSAT solvers, such as akmaxsat [7], maxsatz-2013 [12], WPM1-2013 [1], on the DIMACS benchmark and the BHOSLIB benchmark.

MaxWClq is implemented in C. Both MaxWClq and Cliquer are compiled using gcc 4.8.1 with option "-O2". All experiments are running on a machine with Intel(R) Core2 Duo CPU E8400 @$3.0G$ and $2G$ RAM under Ubuntu 13.10. The cut-off time is $3600s$ for each graph. For random instances, 50 graphs are generated for each case and the average runtime of 50 graphs is considered the runtime for the case.

Table 2. Runtime in sec on DIMACS and BHOSLIB. $|V|$ stands for vertex number, W for the minimal weight, D for density. '-' means cannot be solved within 3600s. Instances solved within 10s by both MWCP solvers or not solved by neither within 3600s are omitted.

Instance	$\|V\|$	D	W	Cliq -uer	MaxW -Clq	akmax -sat	maxs -atz
brock400_1	400	0.74	3422	283	**162**	-	-
brock400_2	400	0.74	3350	398	**157**	-	-
brock400_3	400	0.74	3471	317	**133**	-	-
brock400_4	400	0.74	3626	313	**101**	-	-
brock800_1	800	0.64	3121	**1633**	1725	-	-
brock800_2	800	0.65	3043	**1690**	2441	-	-
brock800_3	800	0.64	3076	**1785**	1874	-	-
brock800_4	800	0.64	2971	**2099**	2501	-	-
C250.9	250	0.89	5092	-	**60.1**	423	-
DSJC1000.5	1000	0.50	2186	**33.4**	99.1	-	-
gen200_p0.9_44	200	0.90	5043	798	**9.64**	39.3	-
gen200_p0.9_55	200	0.90	5416	2016	**3.79**	17.1	-
hamming10-2	1024	0.99	50512	1924	-	127	**12.4**
p_hat1000-2	1000	0.49	5777	-	**3511**	-	-
p_hat300-3	300	0.74	3774	19.8	**3.13**	340	430
p_hat500-3	500	0.75	5375	-	**1254**	-	-
p_hat700-2	700	0.49	5290	196	**66.4**	-	-
san200_0.7_1	200	0.70	3370	23.3	**0.04**	28.8	14.6
san200_0.7_2	200	0.70	2422	464	**0.02**	-	3493
san200_0.9_1	200	0.90	6825	-	**0.31**	1.75	1.63
san200_0.9_2	200	0.90	6082	270	**2.11**	11.2	13.5
san200_0.9_3	200	0.90	4748	2730	**21.2**	123	175
san400_0.5_1	400	0.50	1455	88.1	**0.08**	-	-
san400_0.7_1	400	0.70	3941	-	**5.72**	-	-
san400_0.7_2	400	0.70	3110	-	**8.32**	-	-
san400_0.7_3	400	0.70	2771	-	**10.0**	-	-
san400_0.9_1	400	0.90	9776	-	**1714**	-	-
sanr200_0.9	200	0.89	5126	1339	**8.67**	62.7	106
sanr400_0.7	400	0.70	2992	32.3	**30.2**	-	-
Total No. of DIMACS instances: 80				51	**58**	36	37
frb30-15-1	450	0.82	2990	-	**405**	-	-
frb30-15-2	450	0.82	3006	1188	**54.0**	-	-
frb30-15-3	450	0.82	2995	-	**265**	-	-
frb30-15-4	450	0.82	3032	-	**358**	-	-
frb30-15-5	450	0.82	3011	-	**83.6**	-	-
Total No. of BHOSLIB instances: 5				1	**5**	0	0

Table 1 shows the performances of MaxWClq and Cliquer on random graphs. MaxWClq is faster than Cliquer on 10 (10/12) cases. MaxWClq also dominates Cliquer completely for random graphs with density $D \geq 0.7$. For the cases with $D \geq 0.9$, MaxWClq is at least 30 times faster than Cliquer and MaxWClq outperforms Cliquer even by $1000X$ speed up for case (150,0.95). Furthermore, MaxW-

[6] http://users.tkk.fi/pat/cliquer.html

Clq solves all graphs from case (200, 0.95) and case (300,0.9), which cost 70.75s and 1205s on average respectively, while Cliquer solves none of them. MaxWClq is competitive with Cliquer on the only two cases where Cliquer is faster.

Table 2 presents the experimental results on the DIMACS benchmark and the BHOSLIB benchmark. All 80 instances of the DIMACS benchmark are used. MaxWClq solves 58 instances of them, while Cliquer solves 51 instances. For 29 instances displayed in Table 2, MaxWClq is faster than Cliquer on 23 instances. Furthermore, MaxWClq solves 8 instances which Cliquer cannot finish before the cut-off time, 5 of which only cost MaxWClq less than 100s. *hamming10-2* is the only instance that MaxWClq cannot solve but Cliquer can. For graphs with density $D \geq 0.7$, MaxWClq dominates Cliquer completely except *hamming10-2*. For the 20 instances both algorithms solve within the cut-off time, Cliquer is slightly faster than MaxWClq on 5 instances, but MaxWClq is faster than Cliquer by a large margin on 15 instances (and is even more than 50 times faster than Cliquer on 8 instances among them). Since the BHOSLIB benchmark is much harder, both MaxWclq and Cliquer can only solve the 5 smallest instances with 450 vertices, MaxWClq solves all of them while Cliquer solves only one.

Table 2 also shows that it is inefficient to solve MWCP with MaxSAT solvers directly. For 29 instances displayed in Table 2, MaxWClq is faster than all MaxSAT solvers on all instances except *h*amming10-2. For all 80 instances from the DIMACS benchmark, MaxWClq solves 58 of them, while akmaxsat solves 36 instances and maxsatz solves 37 instances. WPM1-2013 solves only 3 of them (*h*amming6-2, *h*amming8-2, *h*amming10-2), which is not displayed in the table. It is also interesting that MaxSAT solvers can solve *h*amming10-2 effectively, which is hard for both MaxWClq and Cliquer. None of the MaxSAT solvers can solve BHOSLIB instances.

We conclude that MaxWClq outperforms Cliquer on most of instances from the DIMACS benchmark, the BHOSLIB benchmark as well as random graphs. It is competitive with Cliquer on other instances except $hamming10-2$. Meanwhile, MaxWClq is extremely powerful for dense and hard graphs.

7 CONCLUSION

SAT and MaxSAT are powerful tools to solve combinatorial optimization problems. MaxSAT reasoning has been proved to be very effective for MCP. Since MaxSAT encoding and MaxSAT reasoning used in MCP are based on the assumption that each vertex has the same weight, they cannot be directly used to deal with weighted graph. We introduce an approach to encode a weighted graph into a specific weighted MaxSAT and two new simplification rules to split a weighted vertex, so that more conflicts can be detected. Then an algorithm called *Top-k* failed literal detection is proposed to compute a tight upper bound for MWC using the two simplification rules and MaxSAT reasoning. Finally, an exact branch-and-bound solver named MaxWClq is implemented using the new upper bound. Experimental results on commonly used benchmarks and random graphs show that MaxWClq outperforms other state-of-the-art solver for MWC. In particular, MaxWClq is surprisingly powerful for dense and relatively hard graphs.

In the future, we plan to integrate more MaxSAT and MC solving techniques into MaxWClq to solve large-scale and harder instances. It is also interesting to use MaxWClq to solve instances from real-world applications like combinatorial auctions.

ACKNOWLEDGEMENTS

We would like to thank Jichang Zhao, Ge Zheng and anonymous reviewers for their helpful comments and suggestions. This research was partly supported by the National 863 Program of China (No. 2012AA011005), Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20111102110019) and the Chinese State Key Laboratory of Software Development Environment Open Fund (Grant No. SKLSDE-2012KF-07).

REFERENCES

[1] C. Anstegui, M. L. Bonet, and J. Levy, 'Solving (weighted) partial maxsat through satisfiability testing', in *Theory and Applications of Satisfiability Testing-SAT 2009*, pp. 427–440, (2009).
[2] U. Benlic and J. K. Hao, 'Breakout local search for maximum clique problems', *Computers & Operations Research*, **40**, 192–206, (2013).
[3] S. Cai, K. Su, C. Luo, and A. Sattar, 'Numvc: An efficient local search algorithm for minimum vertex cover', *J. Artif. Intell. Res.(JAIR)*, **46**, 687–716, (2013).
[4] S. Cai, K. Su, and A. Sattar, 'Local search with edge weighting and configuration checking heuristics for minimum vertex cover', *Artificial Intelligence*, **175(9)**, 1672–1696, (2011).
[5] T. Fahle, 'Simple and fast: Improving a branch-and-bound algorithm for maximum clique', in *Proc. of ESA-2002*, pp. 485–498, (2002).
[6] J. Konc and D. Janezic, 'An improved branch and bound algorithm for the maximum clique problem', *Communications in Mathematical and in Computer Chemistry*, **58**, 569–590, (2007).
[7] A. Kuegel, 'Improved exact solver for the weighted max-sat problem', in *Workshop Pragmatics of SAT*, volume 436, (2010).
[8] D. Kumlander, '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, (2004).
[9] J. Larrosa, F. Heras, and S. de Givry, 'A logical approach to efficient max-sat solving', *Artificial Intelligence*, **172**, 204–233, (2008).
[10] C. M. Li and Anbulagan A., 'Heuristics based on unit propagation for satisfiability problems', in *Proc. of IJCAI'97*, volume 1, pp. 366–371, (1997).
[11] C. M. Li, F. Manyà, and J. Planes, 'Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers', in *Proc. of CP'05*, volume 3709, pp. 403–414. LNCS, (2005).
[12] C. M. Li, F. Manyà, and J. Planes, 'New inference rules for max-sat', *Journal of Articial Intelligence Research*, **30**, 321–359, (2007).
[13] C. M. Li and Z Quan, 'An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem', in *Proc. of AAAI-2010*, pp. 128–133, (2010).
[14] C.M. Li, Z. Fang, and K. Xu, 'Combining maxsat reasoning and incremental upper bound for the maximum clique problem', in *Proc. of ICTAI-2013*, pp. 939–946, (2013).
[15] P.R. Östergård, 'A new algorithm for the maximum-weight clique problem', *Nordic Journal of Computing*, **8**, 424–436, (2001).
[16] P.R. Östergård, 'A fast algorithm for the maximum clique problem', *Discrete Applied Mathematics*, **120**, 197–207, (2002).
[17] W. Pullan, 'Approximating the maximum vertex/edge weighted clique using local search', *Journal of Heuristics*, **19**, 117–134, (2008).
[18] J. C. Régin, 'Solving the maximum clique problem with constraint programming', in *Proc. of CPAIOR-2003*, pp. 634–648, (2003).
[19] E. Tomita and T. Seki, 'An efficient branch-and-bound algorithm for finding a maximum clique', in *Proc. Discrete Mathematics and Theoretical Computer Science*, volume 2731, pp. 278–289, (2003).
[20] Q. Wu, J. K. Hao, and F. Glover, 'Multi-neighborhood tabu search for the maximum weight clique problem', *Annals of Operations Research*, **196**, 611–634, (2012).
[21] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre, 'Random constraint satisfaction: Easy generation of hard (satisfiable) instances', *Artificial Intelligence*, **171**, 514–534, (2007).
[22] K. Xu and W. Li, 'Exact phase transitions in random constraint satisfaction problems', *J.Artif. Intell. Res. (JAIR)*, **12**, 93–103, (2000).
[23] K. Yamaguchi and S. Masuda, 'A new exact algorithm for the maximum weight clique problem', in *Proc. of ITC-CSCC'08*, pp. 317–320, (2008).