The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

A Two-Stage MaxSAT Reasoning Approach for the Maximum Weight Clique Problem

Hua Jiang,[1] **Chu-Min Li,**[2*] **Yanli Liu,**[3] **Felip Manyà**[4]
[1]College of Mathematics & Computer Science, Jianghan University, China
[2]MIS, Université de Picardie Jules Verne, France
[3]Huazhong University of Science and Technology, China
[4]Artificial Intelligence Research Institute (IIIA, CSIC), Spain
jianghua.hgd@gmail.com; chu-min.li@u-picardie.fr; yanlil2008@163.com; felip@iiia.csic.es

Abstract

MaxSAT reasoning is an effective technology used in modern branch-and-bound (BnB) algorithms for the Maximum Weight Clique problem (MWC) to reduce the search space. However, the current MaxSAT reasoning approach for MWC is carried out in a blind manner and is not guided by any relevant strategy. In this paper, we describe a new BnB algorithm for MWC that incorporates a novel two-stage MaxSAT reasoning approach. In each stage, the MaxSAT reasoning is specialised and guided for different tasks. Experiments on an extensive set of graphs show that the new algorithm implementing this approach significantly outperforms relevant exact and heuristic MWC algorithms in both small/medium and massive real-world graphs.

Introduction

In a vertex-weighted graph $G = (V, E, w)$, where V is the set of vertices and E is the set of edges, the weight function w assigns a positive integer, called *weight*, to each vertex. A *clique* C is a subset of V in which every two vertices are adjacent in G. The size of a clique C is its cardinality. The *Maximum Clique Problem* (MC) is to find a clique of maximum size in G. The weight of a clique C is defined to be the total weight of vertices in C. The *Maximum Weight Clique Problem* (MWC), an important generalization of MC, is to find a clique of maximum weight in G, and its weight is denoted by $\omega_v(G)$.

MC and MWC are NP-hard problems (Garey and Johnson 1979) with applications in areas as diverse as coding theory (Zhian et al. 2013), protein structure prediction (Mascia et al. 2010), combinatorial auctions (Wu and Hao 2015b), computer vision (Zhang, Javed, and Shah 2014) and genomics (Butenko and Wilhelm 2006). Given their practical importance, considerable efforts have been devoted to develop both exact and heuristic algorithms for them.

There exist a remarkable number of algorithms for MC. We highlight the exact branch-and-bound (BnB) algorithms described in (Tomita et al. 2010; Li and Quan 2010; San et al. 2011; Jiang, Li, and Manyà 2016; Li, Jiang, and Manyà 2017), and the heuristic local search algorithms described in (Wu and Hao 2013; Benlic and Hao 2013; Pullan, Mascia, and Brunato 2011). See (Wu and Hao 2015a) for a review on MC algorithms. Compared with the number of algorithms for MC, there are relatively fewer algorithms available for MWC. This is partially due to the fact that MWC is generally harder to solve than MC because of the different weight values of the vertices (Cai and Lin 2016; Jiang, Li, and Manyà 2017).

In this paper, we focus on algorithms for MWC. The most efficient heuristic algorithms for MWC are FastWClq (Cai and Lin 2016), MN/TS (Wu, Hao, and Glover 2012), LSCC+BMS (Wang, Cai, and Yin 2016), ReTS (Zhou, Hao, and Goëffon 2017) and RRWL (Fan et al. 2017). The most representative exact algorithms for MWC are Cliquer (Ostergard 2002), Kumlander's algorithm (Kumlander 2008), VCTable (Shimizu et al. 2012), OTClique (Shimizu et al. 2013), MWCLQ (Fang, Li, and Xu 2016) and WLMC (Jiang, Li, and Manyà 2017). Among them, we identify MWCLQ and WLMC as two of the most competitive algorithms. Both implement the BnB scheme and apply MaxSAT reasoning to improve the upper bound estimation. To our best knowledge, MWCLQ is specially good in small/medium dense graphs while WLMC exhibits the best performance in massive sparse graphs.

MaxSAT reasoning has been proven to be effective to reduce the search space in BnB algorithms for MWC. The current MaxSAT reasoning implemented in MWCLQ and WLMC relies on an upper bound (UB$_{IS}$) derived from the computation of an independent set (IS) partition of the vertices of the graph. However, as stated in (Jiang, Li, and Manyà 2017), UB$_{IS}$ is very conservative. Moreover, the MaxSAT reasoning for improving UB$_{IS}$ implemented in both MWCLQ and WLMC is carried out in a brute-force manner: It repeatedly and blindly applies unit IS propagation to detect disjoint subsets of conflicting ISs.

To overcome the conservativeness of UB$_{IS}$ and improve the efficiency of MaxSAT reasoning on BnB algorithms, we propose a novel two-stage MaxSAT reasoning approach to reducing the search space. In each stage, MaxSAT reasoning is specialised and guided for different tasks. As a result, we develop a new MWC algorithm called TSM-MWC.

The conducted experiments on an extensive set of instances show that the two-stage MaxSAT reasoning approach is very effective on reducing the search space, and TSM-MWC greatly outperforms relevant MWC algorithms in both small/medium and massive real-world graphs. To our best

*Corresponding author

knowledge, this is the first exact algorithm that reaches the best performance in both types of graphs.

The paper is organized as follows: Section 2 gives some basic graph definitions and reviews previous MaxSAT reasoning approaches for MWC. Section 3 presents the two-stage MaxSAT reasoning approach and the algorithm TSM-MWC. Section 4 reports on the empirical results. Section 5 gives the conclusions.

Preliminaries

Let $G = (V, E, w)$ be a vertex-weighted undirected graph, where $V = \{v_1, \ldots, v_n\}$ is a set of n vertices, E is a set of m edges, and w is a weight function that assigns to each vertex v_i of V a non-negative integer $w(v_i)$ representing its weight. We also use $v_i^{w(v_i)}$ to represent that vertex v_i has weight $w(v_i)$. The density of G is $2m/(n(n-1))$. Two vertices v_i and v_j of V are adjacent, or neighbors, if $(v_i, v_j) \in E$. The set of neighbors of a vertex v_i in G is denoted by $\Gamma(v_i) = \{v_j | (v_i, v_j) \in E\}$. The cardinality of $\Gamma(v_i)$ is the degree of v_i. The subgraph of G induced by the subset V' of V, denoted by $G[V']$, is defined as $G[V'] = (V', E', w)$, where $E' = \{(v_i, v_j) \in E | v_i, v_j \in V'\}$. The maximum weight in V', $\max_{v_i \in V'}(w(v_i))$, is denoted by $w^*(V')$, and $w^*(\emptyset) = 0$. A clique in $G = (V, E, w)$ is a subset C of V such that every two vertices in C are adjacent. The weight of C is $w(C) = \sum_{v_i \in C} w(v_i)$. An independent set (IS) of G is a subset D of V in which no two vertices are adjacent. An IS partition of G is a partition of the vertices of V into ISs such that each vertex belongs to exactly one IS.

Let $S = \{v_1^{w(v_1)}, v_2^{w(v_2)}, \ldots, v_r^{w(v_r)}\}$ be a subset of vertices of V with $w(v_1) \geq \cdots \geq w(v_k) > \beta \geq w(v_{k+1}) \geq \cdots \geq w(v_r)$, where β is an integer. The weight splitting operation $split(S, \beta)$ returns the sets $S' = \{v_1^{\beta}, v_2^{\beta}, \ldots, v_k^{\beta}, v_{k+1}^{w(v_{k+1})}, \ldots, v_r^{w(v_r)}\}$ and $S'' = \{v_1^{w(v_1)-\beta}, v_2^{w(v_2)-\beta}, \ldots, v_k^{w(v_k)-\beta}\}$. In other words, for each vertex $v_i \in S$ with $w(v_i) > \beta$, $split(S, \beta)$ splits the weight $w(v_i)$ into β and $w(v_i) - \beta$.

MaxSAT reasoning for MWC

To solve MWC with a BnB algorithm, it is crucial to compute a tight upper bound (UB) of $\omega_v(G)$ for a vertex-weighted graph $G = (V, E, w)$. Given an IS partition $\Pi = \{D_1, D_2, \ldots, D_r\}$ of G, the UB of $\omega_v(G)$ based on ISs is defined to be $\text{UB}_{IS} = \sum_{i=1}^{r} w^*(D_i)$, because a clique of G contains at most one vertex from each IS. However, UB_{IS} is very conservative. It is tight only in the very special case where there is a maximum weight clique that contains the most weighted vertex of each IS in Π.

A subset of k ISs that cannot form a clique of size k is said *conflicting* (the k ISs are also said *conflicting*). A special case is $k = 2$: two ISs S_1 and S_2 are conflicting iff $S_1 \cup S_2$ is an IS. If $\{S_1, S_2, \ldots, S_k\}$ is a conflicting subset of ISs of Π and $\beta = \min(w^*(S_1), w^*(S_2), \ldots, w^*(S_k))$, then $\text{UB}_{IS} - \beta$ is an improved UB of $\omega_v(G)$. This follows from the fact that, for every possible clique C, there is at least one IS S in $\{S_1, S_2, \ldots, S_k\}$ such that $C \cap S = \emptyset$.

To increase the number of disjoint conflicting subsets of ISs, we split the ISs of the already detected conflicting subsets of ISs. Concretely, each S_i $(1 \leq i \leq k)$ of a conflicting subset $\{S_1, S_2, \ldots, S_k\}$ of ISs can be split, using the operation $split(S_i, \beta)$, into two ISs: S_i' and S_i''. Note that the subset $\Pi_1 = \{S_1', S_2', \ldots, S_k'\}$, where $w^*(S_1') = w^*(S_2') = \cdots = w^*(S_k') = \beta$, is conflicting. Additional conflicting subsets of ISs, which are disjoint with $\{S_1', S_2', \ldots, S_k'\}$, can be identified in $(\Pi \setminus \{S_1, S_2, \ldots, S_k\}) \cup \{S_1'', S_2'', \ldots, S_k''\}$.

Since each conflicting subset of k ISs implies the splitting of k existing ISs, we finally obtain a set of ISs $\{D_1, D_2, \ldots, D_q\}$ that is the union $\Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_{p-1} \cup \Pi_p$, and each Π_i, $1 \leq i \leq p-1$, is a conflicting subset of ISs. Since each vertex v of G belongs to at most one of the ISs of Π_i, we define a weight function $w_i(v)$ for each Π_i as follows: $w_i(v)$ is 0 if v does not belong to any IS of Π_i; otherwise $w_i(v)$ is the weight of v in the IS of Π_i in which v appears. Obviously, each v of G can belong to several subsets of $\Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_{p-1} \cup \Pi_p$ and it must hold that $w(v) = \sum_{i=1}^{p} w_i(v)$.

Each Π_i induces a graph G_i that has the same set of vertices and the same set of edges as G, but with a different weight function $w_i(v)$. Let C be a clique of G, then $w(C) = \sum_{v \in C} w(v) = \sum_{v \in C} \sum_{i=1}^{p} w_i(v) = \sum_{i=1}^{p} \sum_{v \in C} w_i(v)$. Note that C is also a clique in each G_i and G_i can be partitioned into a set of ISs $\{S_1', S_2', \ldots, S_{k_i}'\}$. We have that $\sum_{v \in C} w_i(v) \leq \sum_{j=1}^{k_i} w_i^*(S_j') - \beta_i$ for $i < p$, because the ISs are conflicting. So, $w(C) \leq \sum_{i=1}^{p-1} (\sum_{j=1}^{k_i} w_i^*(S_j') - \beta_i) + \sum_{j=1}^{k_p} w_p^*(S_j') = \sum_{i=1}^{p} \sum_{j=1}^{k_i} w_i^*(S_j') - \sum_{i=1}^{p-1} \beta_i = \sum_{i=1}^{q} w_i^*(D_i) - \sum_{i=1}^{p-1} \beta_i$.

The improved upper bound UB_{MaxSAT}, obtained by identifying and splitting conflicting ISs, is defined to be $\sum_{i=1}^{q} w_i^*(D_i) - \sum_{i=1}^{p-1} \beta_i$. It is implemented in the two best performing exact MWC algorithms, MWCLQ and WLMC.

Let C^* be the clique of the greatest weight found so far in G. If UB_{MaxSAT} is not sufficient for pruning (i.e., $\text{UB}_{MaxSAT} > w(C^*)$), further search is required to find a better clique in G. In this case, the effort spent to compute UB_{MaxSAT} in MWCLQ is lost. To overcome this drawback, the algorithm WLMC first identifies a subset A of V in such a way that $G[A]$ has an IS partition Π and its UB_{IS} is not greater than $w(C^*)$. Let $B = V \setminus A = \{b_1, b_2, \ldots, b_{|B|}\}$. If $B = \emptyset$, the search is pruned, because G does not contain any clique of weight greater than $w(C^*)$. Otherwise, WLMC successively adds b_i as a unit IS $\{b_i\}$ to Π for $i = |B|, |B| - 1, \ldots, 1$. For each b_i, WLMC employs MaxSAT reasoning to compute UB_{MaxSAT} for $G[A \cup \{b_i\}]$. If the computed UB_{MaxSAT} is not greater than $w(C^*)$, b_i is removed from B and added to A, because a clique of weight greater than $w(C^*)$ cannot be found in $G[A \cup \{b_i\}]$. In this way, the number of vertices of B is greatly reduced by MaxSAT reasoning. Finally, WLMC only needs to branch on the vertices remaining in B to find a clique better than C^*.

Note that the computation of UB_{MaxSAT} in MWCLQ and WLMC is based on UB_{IS}, which is very conservative. Moreover, MWCLQ and WLMC identify a conflicting subset of ISs by repeatedly and blindly applying *unit clause propaga-*

tion or *unit IS propagation*, which is a brute force approach and might impede the computation of a tighter UB.

TSM-MWC: A BnB MWC Algorithm with Two-stage MaxSAT Reasoning

In this section, we describe the new BnB MWC algorithm TSM-MWC. We first present the basic BnB search procedure that TSM-MWC implements, and then describe a novel two-stage MaxSAT reasoning procedure (TSM) to reduce the number of branching vertices of the search procedure. Finally, we define TSM-MWC, which combines an efficient preprocessing and the novel two-stage MaxSAT reasoning.

The basic BnB search procedure

Algorithm 1 presents the basic BnB search procedure that TSM-MWC implements. Given a graph $G = (V, E, w)$, a vertex ordering O over V, a growing clique C, a candidate set of vertices P in which every vertex is adjacent to every vertex in C, and the clique C^* of the greatest weight found so far in G, the algorithm calls the *GetBranches* function to partition P into A and B in such a way that $\omega_v(G[A]) \leq w(C^*) - w(C)$ and $B = V \setminus A = \{b_1, b_2, \ldots, b_{|B|}\}$ is the returned set of branching vertices. If B is empty, the current search is pruned, because the clique C cannot be grown to a clique of weight greater than $w(C^*)$ with the vertices of P. Otherwise, the algorithm recursively branches on b_i, for $i = |B|, |B| - 1, \ldots, 1$, to search for a clique containing b_i of weight greater than $w(C^*)$ in $G[\Gamma(b_i) \cap (\{b_{i+1}, \ldots, b_{|B|}\} \cup A)]$. If the initial call to the algorithm is SearchMWC$(G, V, O, \emptyset, \emptyset)$, it returns a maximum weight clique of G after exploring the whole search space.

The crucial component of Algorithm 1 is the GetBranches function. The smaller the cardinality of the set of branching vertices B returned by GetBranches, the lower the number of branches that need to be explored to find an optimal solution. So, in the sequel, we will focus on developing a better GetBranches function based on MaxSAT reasoning.

Algorithm 1: SearchMWC(G, P, O, C, C^*), a generic BnB algorithm for MWC

Input: $G = (V, E, w)$, a candidate set P, an ordering O over P, a growing clique C, and the greatest weight clique C^* found so far in G.
Output: $C \cup C'$, where C' is a maximum weight clique of $G[P]$, if $w(C \cup C') > w(C^*)$; C^* otherwise.

begin
  if $P = \emptyset$ **then return** C ;
  $B \leftarrow GetBranches(G[P], w(C^*) - w(C), O)$;
  if $B = \emptyset$ **then return** C^* ;
  Let $B = \{b_1, \ldots, b_{|B|}\}$, $b_1 < \cdots < b_{|B|}$ w.r.t. O;
  $A \leftarrow V \setminus B$;
  for $i := |B|$ *downto 1* **do**
    $P' \leftarrow \Gamma(b_i) \cap (\{b_{i+1}, \ldots, b_{|B|}\} \cup A)$;
    $C_1 \leftarrow SearchMWC(G, P', O, C \cup \{b_i\}, C^*)$;
    if $w(C_1) > w(C^*)$ **then** $C^* \leftarrow C_1$;
  return C^*;

Two-stage MaxSAT Reasoning to Minimize the Number of Branches

We describe a two-stage MaxSAT reasoning approach that considers in priority conflicting subsets containing two or three ISs, allowing to reduce more branches than previous MaxSAT reasoning approaches.

Stage 1. Given a vertex-weighted graph $G = (V, E, w)$, a lower bound t of $\omega_v(G)$ and a vertex ordering $O : v_1 < v_2 < \cdots < v_n$, Algorithm 2 defines a simplified MaxSAT reasoning approach to computing an initial set of branching vertices B and a set Π of ISs of $V \setminus B$ so that $\omega_v(G[V \setminus B]) \leq \sum_{D \in \Pi} w^*(D) \leq t$. Thus, to search for a clique of weight greater than t, it suffices to branch on the vertices of B.

Initially, B and Π are empty. For $i = n$ to 1, if every IS of Π contains at least one neighbor of v_i, Algorithm 2 adds the new IS $\{v_i\}$ to Π if $\sum_{D \in \Pi} w^*(D) + w(v_i) \leq t$ (lines 10–11). Otherwise, the algorithm splits the weight $w(v_i)$ among some ISs $S_1, S_2, \ldots, S_k$ of Π not containing any neighbor of v_i by applying the following lemma.

Lemma 1. Let $G = (V, E, w)$ be a vertex-weighted graph, let $\Pi = \{D_1, D_2, \ldots, D_r\}$ be a set of ISs, let $V(\Pi)$ be the set of vertices occurring in Π, let v_i be a vertex of $V \setminus V(\Pi)$, and let $S_1, S_2, \ldots, S_k$ be ISs of Π not containing any neighbor of v_i (i.e., conflicting with $\{v_i\}$). Then, $\sum_{D \in \Pi} w^*(D) + \max(w(v_i) - \sum_{j=1}^{k} w^*(S_j), 0)$ is an upper bound of $\omega_v(G[V(\Pi) \cup \{v_i\}])$.

Proof. Let C be a maximum weight clique in $G(V(\Pi) \cup \{v_i\})$. If $v_i \in C$, then C cannot contain any vertex from S_1, S_2, …, S_k and $\sum_{D \in \Pi \setminus \{S_1, S_2, \ldots, S_k\}} w^*(D) + w(v_i) = \sum_{D \in \Pi} w^*(D) + w(v_i) - \sum_{j=1}^{k} w^*(S_j)$ is an upper bound of $\omega_v(G[V(\Pi) \cup \{v_i\}])$. If $v_i \notin C$, $\sum_{D \in \Pi} w^*(D)$ is an upper bound of $\omega_v(G[V(\Pi) \cup \{v_i\}])$. Hence, $\sum_{D \in \Pi} w^*(D) + \max(w(v_i) - \sum_{j=1}^{k} w^*(S_j), 0)$ is an upper bound of $\omega_v(G[V(\Pi) \cup \{v_i\}])$ in both cases. □

Let $ub = \sum_{D \in \Pi} w^*(D)$ and let $\delta = w(v_i)$. If $ub + \max(\delta - w^*(S_1), 0) \leq t$, then Algorithm 2 inserts v_i^{δ} into S_1 and updates ub to $ub + \max(\delta - w^*(S_1), 0)$, because $G[V(\Pi) \cup \{v_i\}]$ cannot contain any clique of weight greater than t in this case according to Lemma 1. Otherwise, Algorithm 2 inserts $v_i^{w^*(S_1)}$ into S_1 in order not to increase $w^*(S_1)$, updates δ to $\delta - w^*(S_1)$, and tries to insert v_i^{δ} into S_2, and so on (lines 16–23). In a word, Algorithm 2 splits $v_i^{w(v_i)}$ into $v_i^{w^*(S_1)}, v_i^{w^*(S_2)}, \ldots, v_i^{w^*(S_{k'-1})}, v_i^{\delta}$, where $\delta = w(v_i) - \sum_{j=1}^{k'-1} w^*(S_j)$ and k' is the smallest integer such that $k' \leq k$ and $ub + \max(w(v_i) - \sum_{j=1}^{k'} w^*(S_j), 0) \leq t$. It adds $v_i^{w^*(S_j)}$ to S_j for $j = 1$ to $k' - 1$, inserts v_i^{δ} into $S_{k'}$, and updates ub to $ub + \max(w(v_i) - \sum_{j=1}^{k'} w^*(S_j), 0)$. If such a k' does not exist, the algorithm inserts v_i into B and restores Π to the values it had before considering v_i (line 24). Finally, it returns the set B, the set Π of ISs on $V \setminus B$ and an upper bound ub of $\omega_v(G[V \setminus B])$.

Algorithm 2: BinaryMaxSAT(G, t, O)

Input: $G = (V, E, w)$, an integer t and an ordering O
Output: a set B of branching vertices, a set Π of ISs and an upper bound ub of $\omega_v(G[V \setminus B])$

begin
  $ub \leftarrow 0$; $B \leftarrow \emptyset$; $\Pi \leftarrow \emptyset$; /* Π is a set of ISs*/
  Let $V = \{v_1, \dots, v_n\}$, $v_1 < \dots < v_n$ w.r.t. O;
  for $i := n$ *downto* 1 **do**
    $\Pi' \leftarrow \Pi$, $\delta \leftarrow w(v_i)$;
    remove all non-neighbors of v_i from their ISs;
    if *no empty IS is produced* **then**
      restore all removed vertices into their ISs;
      if $ub + \delta \leq t$ **then**
        create a new IS $D = \{v_i^{\delta}\}$;
        $\Pi \leftarrow \Pi \cup \{D\}$, $ub \leftarrow ub + \delta$;
      else $B \leftarrow B \cup \{v_i\}$;
    else
      let $S_1, S_2, \dots, S_k$ be the empty ISs;
      restore all removed vertices into their ISs;
      for $j := 1$ *to* k **do**
        if $ub + \max(\delta - w^*(S_j), 0) \leq t$ **then**
          $S_j \leftarrow S_j \cup \{v_i^{\delta}\}$;
          $ub \leftarrow ub + \max(\delta - w^*(S_j), 0)$;
          $\delta \leftarrow 0$; **break**;
        else
          $S_j \leftarrow S_j \cup \{v_i^{w^*(S_j)}\}$;
          $\delta \leftarrow \delta - w^*(S_j)$;
      if $\delta > 0$ **then** $\Pi \leftarrow \Pi'$, $B \leftarrow B \cup \{v_i\}$;
  return (B, Π, ub);

The key of Algorithm 2 is the identification of the ISs $S_1, \dots, S_k$, which are conflicting with $\{v_i\}$, and the splitting of $w(v_i)$ among these ISs, which is referred to as *binary MaxSAT reasoning*, because $\{S_j, \{v_i\}\}$ ($1 \leq j \leq k$) is a binary conflicting subset of ISs. Note that Π is not an IS partition of $G[V \setminus B]$ in the strict sense, because a vertex can belong to several ISs of Π.

Let θ be the greatest weight among the vertices of V. Each vertex $v \in V$ has at most $w(v)$ occurrences in Π. So, the total number of vertices in Π is in $O(\theta \times |V|)$. The main cost of inserting a vertex v into Π is the identification of the ISs $S_1, S_2, \dots, S_k$ not containing any neighbor of v. So, the time complexity of inserting a vertex into Π is in $O(\theta \times |V|)$, and the time complexity of Algorithm 2 is in $O(\theta \times |V|^2)$.

Stage 2. This stage is implemented in Algorithm 3. Its aim is to further reduce the set B of branching vertices returned by Algorithm 2 together with the set of ISs Π on $V \setminus B$. To remove a vertex b from B, the algorithm checks if $\omega_v(G[V(\Pi) \cup \{b\}]) \leq t$ to prove that branching on b is not necessary. For this purpose, it identifies conflicting subsets of ISs in $\Pi \cup \{\{b\}\}$ in an ordered way: Firstly, the conflicting subsets containing 2 ISs are identified; secondly, the conflicting subsets containing 3 ISs; and finally, the conflicting subsets containing more than 3 ISs. We refer to this MaxSAT reasoning approach as *ordered MaxSAT reasoning*.

Algorithm 3: OrderedMaxSAT(G, t, O,B, Π, ub)

Input: $G = (V, E, w)$, an integer t, an ordering O over V, a subset B of V, a set Π of ISs on $V \setminus B$, and an upper bound ub of $\omega_v(G[V \setminus B])$
Output: a set B of branching vertices

1 **begin**
2   Let $B = \{b_1, \dots, b_{|B|}\}$, $b_1 < \dots < b_{|B|}$ w.r.t. O;
3   **for** $i := |B|$ *downto* 1 **do**
4     $\Pi' \leftarrow \Pi$, $\delta \leftarrow w(b_i)$;
5   let $S_1, S_2, \dots, S_k$ be the ISs containing no neighbor of b_i;
6   **for** $j := 1$ *to* k **do**
7       $S_j \leftarrow S_j \cup \{b_i^{w^*(S_j)}\}$; $\delta \leftarrow \delta - w^*(S_j)$;
8   let $U_1, U_2, \dots, U_r$ be the ISs containing exactly one neighbor of b_i;
9   **for** $j := 1$ *to* r **do**
10   let $\Gamma(b_i) \cap U_j = \{u\}$;
11   **if** *there is an IS* D_q *such that* $D_q \cap \Gamma(b_i) \cap \Gamma(u) = \emptyset$ **then**
12         $\beta \leftarrow \min(\delta, w^*(U_j), w^*(D_q))$;
13         $(U'_j, U''_j) \leftarrow split(U_j, \beta)$;
14         $(D'_q, D''_q) \leftarrow split(D_q, \beta)$;
15         $\Pi \leftarrow (\Pi \setminus \{U_j, D_q\}) \cup \{U''_j, D''_q\}$;
16         $\delta \leftarrow \delta - \beta$;
17   **if** $ub + \delta \leq t$ **then break;**
18   **if** $ub + \delta > t$ **then**
19       $\Pi \leftarrow \Pi \cup \{\{b_i^{\delta}\}\}$;
20   **while** *there is a unit IS* $\{v\}$ *in* Π **do**
21   remove the non-neighbors of v from ISs;
22   **if** *there is an empty IS* S_0 **then**
23   let $S_1, \dots, S_p$ be the ISs responsible
24   of removing all the vertices of S_0;
25   restore the removed vertices into ISs;
26           $\beta \leftarrow \min(w^*(S_0), \dots, w^*(S_p))$;
27   **for** *each* S_j *in* $\{S_0, S_1, \dots, S_p\}$ **do**
28             $(S'_j, S''_j) \leftarrow split(S_j, \beta)$;
29           $\Pi \leftarrow (\Pi \setminus \{S_0, \dots, S_p\}) \cup \{S''_0, \dots, S''_p\}$;
30           $\delta \leftarrow \delta - \beta$;
31   **if** $ub + \delta \leq t$ **then break;**
32   **if** $ub + \delta \leq t$ **then**
33       $B \leftarrow B \setminus \{b_i\}$, $ub \leftarrow ub + \delta$;
34   **else** $\Pi \leftarrow \Pi'$;
35   **return** B;

Concretely, let $B = \{b_1, b_2, \dots, b_{|B|}\}$ with $b_1 < b_2 < \dots < b_{|B|}$ w.r.t. the ordering O and let $ub = \sum_{D \in \Pi} w^*(D)$. For each b_i ($1 \leq i \leq |B|$), note that $ub + w(b_i)$ is an upper bound of $\omega_v(G[V(\Pi) \cup \{b_i\}])$. Algorithm 3 improves this upper bound by first identifying a set of ISs $\{S_1, S_2, \dots, S_k\}$ in Π not containing any neighbor of b_i, splitting $w(b_i)$ into $w^*(S_1), w^*(S_2), \dots, w^*(S_k)$, and δ, where $\delta = w(b_i) - \sum_{j=1}^{k} w^*(S_j)$, and inserting $b_i^{w^*(S_j)}$ into S_j ($1 \leq j \leq k$) (lines 6 and 7). After these insertions, the upper bound of $\omega_v(G[V(\Pi) \cup \{b_i\}])$ is improved to $ub + \delta$. Note that $\delta > 0$ because b_i was not entirely inserted into Π in Stage 1.

Then, Algorithm 3 identifies a set of ISs $\{U_1, U_2, \ldots, U_r\}$ containing exactly one neighbor of b_i. For each U_j $(1 \leq j \leq r)$, let u be the unique neighbor of b_i in U_j. The algorithm tries to identify an IS D_q such that $D_q \cap \Gamma(b_i) \cap \Gamma(u) = \emptyset$. Thus, $\{\{b_i^\delta\}, U_j, D_q\}$ is conflicting. Let $\beta = \min(\delta, w^*(U_j), w^*(D_q))$, the algorithm splits b_i^δ into b_i^β and $b_i^{\delta-\beta}$, and U_j (D_q) into U'_j (D'_q) and U''_j (D''_q) using the operation $split(U_j, \beta)$ $(split(D_q, \beta))$ defined in Section Preliminaries. Note that the maximum weight in both U'_j and D'_q is β, and the upper bound of $\omega_v(G[V(\Pi) \cup \{b_i\}])$ is improved to $ub + \delta - \beta$ with the conflicting subset $\{\{b_i^\beta\}, U'_j, D'_q\}$. Note that U'_j and D'_q cannot be used for other improvements and are excluded from Π together with U_j and D_q (line 15). After the improvement, δ is updated to $\delta - \beta$ (line 16).

After working on all ISs in $\{U_1, U_2, \ldots, U_r\}$, if $ub + \delta$ is still greater than t, Algorithm 3 repeatedly detects disjoints conflicting subsets of ISs to improve $ub + \delta$; similarly to the approach implemented in MWCLQ and WLMC. For each detected disjoint conflicting subset of ISs $\{S_0, S_1, \ldots, S_p\}$, let $\beta = \min(w^*(S_0), w^*(S_1), \ldots, w^*(S_p))$. The algorithm splits every S_j $(0 \leq j \leq p)$ into S'_j and S''_j with the operation $split(S_j, \beta)$ and improves $ub + \delta$ by β (lines 26–30). Finally, if the improved UB of $\omega_v(G[V(\Pi) \cup \{b_i\}])$ is not greater than t, b_i is removed from B, because branching on b_i is not necessary for finding a clique of weight greater than t. Otherwise, the algorithm restores Π to the values it had before considering b_i.

The time complexity of Algorithm 3 is dominated by the third part (line 18-31) of the detection of the conflicting subsets containing more than 3 ISs. So, its time complexity is similar to the MaxSAT reasoning approach in WLMC.

The next example illustrates the benefits of the two-stage MaxSAT reasoning approach in reducing the number of branches, and compares it with the standard IS partition approach and the brute-force MaxSAT reasoning in WLMC.

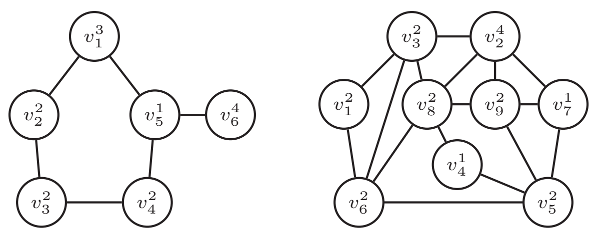

Figure 1: Two graphs for Example 1

Example 1. Let $G = (V, E, w)$ be the left graph (a) of Figure 1, where $v_i^{w_i}$ denotes that the weight of vertex v_i is w_i, let O: $v_6 < v_5 < \cdots < v_1$ be the vertex ordering and let $t = 6$. Using the standard IS partition approach, the vertices $v_1, \ldots, v_5$ can be sequentially inserted into three ISs: $D_1 = \{v_1^3, v_3^2\}, D_2 = \{v_2^2, v_4^2\}$ and $D_3 = \{v_5^1\}$ with $\text{UB}_{IS} = 3 + 2 + 1 = 6 \leq t$. Nevertheless, the last vertex v_6^4 cannot be inserted into D_1 or D_2, because then UB_{IS} would be at least 7, which is greater than t. So, $B = \{v_6^4\}$ is the set of branching vertices computed with the standard IS partition approach. However, with the binary MaxSAT reasoning in Stage 1 (Algorithm 2), D_1 and D_2 are identified, and the algorithm inserts v_6^3 into D_1 and v_6^1 into D_2 so that the computed UB$= 6 + \max\{4 - (3 + 2), 0\} = 6$ is not yet greater than t. As a result, the returned set B of branching vertices for the left graph (a) of Figure 1 is the empty set.

We illustrate the benefits of the ordered MaxSAT reasoning in Stage 2 with the right graph (b) of Figure 1. Let $v_9 < v_8 < \cdots < v_1$ be the vertex ordering O and let $t = 8$. Using the standard IS partition approach, $v_1, \ldots, v_7$ can be partitioned into three ISs: $D_1 = \{v_1^2, v_2^4, v_4^1\}, D_2 = \{v_3^2, v_5^2\}$ and $D_3 = \{v_6^2, v_7^1\}$ with $\text{UB}_{IS} = 4 + 2 + 2 = 8 \leq t$. Nevertheless, v_8^2 and v_9^2 cannot be inserted into any of the three ISs. Hence, the initial set of branching vertices is $B = \{v_8^2, v_9^2\}$. To reduce B, we apply MaxSAT reasoning. First, we add $\{v_8^2\}$ to the set of ISs as a unit IS, and $\text{UB}_{IS} = 10$. The propagation of the unit IS $\{v_8^2\}$ removes v_1^2, v_5^2 and v_7^1 from their ISs, resulting in two new unit ISs: $D_2 = \{v_3^2\}$ and $D_3 = \{v_6^2\}$. With brute-force MaxSAT reasoning, we select $D_2 = \{v_3^2\}$ and propagate it, removing v_4^1 from D_1. We then propagate $D_3 = \{v_6^2\}$, removing v_2^4 from D_1 and making D_1 empty. Since $\{v_8^2\}$, D_2 and D_3 are the reasons for removing v_1^2, v_4^1 and v_2^4 from D_1, respectively, $\{D_1, D_2, D_3, \{v_8^2\}\}$ is a conflicting subset of ISs and UB_{IS} can be improved to $\text{UB}_{IS} - \beta = 10 - 2 = 8 \leq t$, where $\beta = \min\{w^*(D_1), w^*(D_2), w^*(D_3), w^*(\{v_8^2\})\} = 2$. After splitting D_1, D_2, D_3 and $\{v_8^2\}$ with the operation $split(D, \beta)$, the ISs D_2, D_3 and $\{v_8^2\}$ are removed from the partition, because their maximum vertex weight is equal to β, and D_1 is split into $D'_1 = \{v_1^2, v_2^2, v_4^1\}$ and $D''_1 = \{v_2^2\}$ (D'_1 is removed from the partition). After that, we cannot identify any new conflict subset when we add $\{v_9^2\}$ to the set of ISs, because v_9^2 is adjacent to v_2^2 in D''_1. Finally, the returned branching set is $B = \{v_9^2\}$.

However, if we use the ordered MaxSAT reasoning in Stage 2 (Algorithm 3), we can remove v_9^2 from B. Indeed, after propagating the new unit IS $\{v_8^2\}$, we have two new unit ISs: $D_2 = \{v_3^2\}$ and $D_3 = \{v_6^2\}$. According to Algorithm 3, the conflicting subsets containing two or three ISs are detected in priority. Hence, the algorithm identifies the conflicting subset $\{D_1, D_3, \{v_8^2\}\}$, because the neighbors of v_8^2 in D_1 and D_3 are not adjacent, and UB_{IS} is improved to $\text{UB}_{IS} - \beta = 8 \leq t$, where $\beta = \min\{w^*(D_1), w^*(D_3), w^*(\{v_8^2\})\} = 2$. After splitting D_1, D_3 and $\{v_8^2\}$ with the operation $split(D, \beta)$, the remaining ISs are $D''_1 = \{v_2^2\}$ and $D_2 = \{v_3^2, v_5^2\}$. We then add $\{v_9^2\}$ to the set of ISs, increasing UB_{IS} to 10. Now, we can easily detect a new conflicting subset of ISs: $\{D''_1, D_2, \{v_9^2\}\}$, because D_2 does not contain any vertex that is adjacent to both v_2^2 and v_9^2, and UB_{IS} is improved to $\text{UB}_{IS} - 2 = 8 \leq t$. So, v_9^2 is removed from B. Finally, the returned branching set for the right graph (b) of Figure 1 is the empty set.

Note that, in Stage 1, binary MaxSAT reasoning does not remove any IS from the set Π of ISs. So, ordered MaxSAT reasoning can consider all the ISs in Π in Stage 2. This is beneficial for detecting more disjoint conflicting subsets of ISs to reduce the number of branches. That is the rationale behind carrying out MaxSAT reasoning in two stages.

Combining Algorithm 2 and Algorithm 3, we can easily implement the *GetBranches* function in Algorithm 4 to minimize the number of branching vertices.

Algorithm 4: GetBranches(G, t, O)

Input: $G = (V, E, w)$, an integer t and an ordering O
Output: a set B of branching vertices
begin
  $(B, \Pi, ub) \leftarrow BinaryMaxSAT(G, t, O)$;
  if B *is not empty* **then**
    $B \leftarrow OrderedMaxSAT(G, t, O, B, \Pi, ub)$;
  return B;

Algorithm TSM-MWC

Preprocessing is crucial for the efficiency of BnB algorithms for MWC, especially on massive real-world graphs. Thus, to obtain the new exact algorithm TSM-MWC (Algorithm 5), we combine the efficient preprocessing of Algorithm WLMC with Algorithm 1.

Given $G = (V, E, w)$ and a lower bound t of $\omega_v(G)$, the preprocessing procedure *Initialize(G, t)* of WLMC performs three tasks: it derives a vertex ordering O, seeks an initial clique C_0 and reduces the input graph G to a simpler graph G'. Initialize(G, t) computes the vertex ordering $O : v_1 < v_2 < \cdots < v_n$ as follows: Given a copy H of G, it removes the vertex with the smallest degree in H and names it v_1; then, it removes the vertex with the smallest degree in $H[V \setminus \{v_1\}]$ and names it v_2, and so on. After removing the vertices $v_1, v_2, \ldots, v_i$ from H, if the smallest degree in $H[V \setminus \{v_1, v_2, \ldots, v_i\}]$ is $|V|-i-1$, then $V \setminus \{v_1, v_2, \ldots, v_i\}$ becomes the initial clique C_0. If $w(C_0) > t$, $t = w(C_0)$. Initialize(G, t) returns $(C_0, O, G[V'])$, where $V' = \{v \mid w(\{v\} \cup \Gamma(v)) > t\}$.

TSM-MWC calls Initialize(G, t) to preprocess the original graph G (line 2) and all the first level subgraphs $G[P]$ (line 8), and then calls SearchMWC (Algorithm 1) to search for a clique of weight greater than $w(C^*)$ in the reduced subgraph G'' (line 12).

Algorithm 5: TSM-MWC(G), an exact algorithm for MWC

Input: $G = (V, E, w)$
Output: a maximum weight clique C^* of G
1 **begin**
2   $(C_0, O_0, G') \leftarrow Initialize(G, 0)$;
3   $C^* \leftarrow C_0$, $V' \leftarrow$ the vertex set of G';
4   order V' w.r.t. the initial ordering O_0;
5   **for** $i := |V'|$ *downto* 1 **do**
6     $C \leftarrow \{v_i\}$, $P \leftarrow \Gamma(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_{|V'|}\}$;
7   **if** $w(P) + w(C) > w(C^*)$ **then**
8       $(C'_0, O'_0, G'') \leftarrow Initialize(G[P], w(C^*) - w(C))$;
9   **if** $w(C'_0) + w(C) > w(C^*)$ **then**
10        $C^* \leftarrow C'_0 \cup C$;
11      $V'' \leftarrow$ the vertex set of G'';
12      $C_1 \leftarrow SearchMWC(G'', V'', O'_0, C, C^*)$;
13   **if** $w(C_1) > w(C^*)$ **then** $C^* \leftarrow C_1$;
14 **return** C^*;

Empirical Investigation

We empirically evaluated TSM-MWC and compared it with two of the most competitive and recent exact algorithms (also called *solvers*), MWCLQ (Fang, Li, and Xu 2016) and WLMC (Jiang, Li, and Manyà 2017), and FastWClq (Cai and Lin 2016), one of the best heuristic MWC solvers.

TSM-MWC was implemented in C and compiled using GNU gcc -O3. Its source code is available at http://home.mis.u-picardie.fr/~cli/EnglishPage.html. MWCLQ, WLMC and FastWClq were compiled using their Makefiles. Experiments were performed on Intel Xeon CPUs E5-2680 v4@2.40GHz under Linux with 128GB of memory. We considered three datasets:

- **DIMACS graphs:** 80 graphs containing up to 4000 vertices with densities ranging from 0.03 to 0.99.[1]
- **Real-world massive graphs:** 215 real-world sparse graphs from the Network Data Repository, containing up to 66M vertices and 1800M edges.[2]
- **Graphs from MWC practical applications:** There are four groups: The winner determination problem (WDP), error-correcting codes (ECC), kidney-exchange schemes (KES) and the research excellence framework (REF). They contain up to 8900 vertices with densities ranging from 0.04 to 0.98, and were recently recommended in (McCreesh et al. 2017) to evaluate MWC algorithms.[3]

The weights in the first two datasets are assigned as in the most relevant literature. The weights in the third dataset represent real meanings and can be very large; e.g. the vertex weight represents the price of a bid in the WDP graphs, and the paper ranking in the REF graphs.

We first compare TSM-MWC with MWCLQ, WLMC and FastWClq, and then analyze the effect of the two-stage MaxSAT reasoning in TSM-MWC. To compare the heuristic solver FastWClq with the exact solvers, FastWClq solved each graph 10 times with different seeds. The mean time to reach the best solution in each run (avgt), and the best solution found over the 10 runs are reported (best).

Comparison of TSM-MWC with Other Solvers

We solved 80 DIMACS graphs using a cutoff time of 5000s to evaluate TSM-MWC on small/medium dense graphs. The exact solvers TSM-MWC, MWCLQ and WLMC solved 66, 61 and 60 DIMACS graphs, respectively. Table 1 shows the results for the 34 instances resulting of excluding the easy graphs that all the exact solvers solved within 1s, and the hard

[1] available at http://cs.hbg.psu.edu/txn131/clique.html

[2] available at http://networkrepository.com

[3] https://github.com/jamestrimble/max-weight-clique-instances

Table 1: Comparison of runtimes in seconds of TSM-MWC with MWCLQ, WLMC and FastWClq on DIMACS graphs with a cutoff time of 5000 seconds. The best times are in bold. "↓" means that the best solutions found by FastWClq are not optimal.

Instance	$\omega_v(G)$	TSM-MWC	MW-CLQ	W-LWC	FastWClq best	FastWClq avgt
brock400_1	3422	**112.4**	123.6	426.5	3422	188.3
brock400_2	3350	140.9	113.4	542.9	3350	**43.7**
brock400_3	3471	82.9	91.9	356.2	3471	**7.13**
brock400_4	3626	139.4	71.7	595.6	3626	**1.20**
brock800_1	3121	1714	1294	–	3121	**74.17**
brock800_2	3043	2336	1874	–	3043	**317.3**
brock800_3	3076	1930	1378	–	3076	**18.3**
brock800_4	2971	2410	1878	–	2971	**1558**
C250.9	5092	18.6	34.8	83.7	5092	**3.11**
DSJC1000_5	2186	81.1	**75.1**	219.6	2186	98.9
DSJC500_5	1725	1.34	**0.79**	2.98	1725	6.32
gen200_p0.9_44	5043	0.61	6.24	2.78	5043	**0.25**
gen200_p0.9_55	5416	**0.70**	2.43	2.74	5416	11.3
gen400_p0.9_75	8006	**356.6**	–	–	8006	2047
hamming10-2	50512	34.2	841.4	1393	50512	**8.85**
MANN_a27	12283	4.24	–	**1.02**	12258↓	106.9
MANN_a45	34265	1323	–	**357.1**	34121↓	1041
p_hat1000-2	5777	**13.8**	2103	86.2	5777	2748
p_hat1500-1	1619	**2.88**	3.46	5.54	1619	33.8
p_hat1500-2	7360	**660.2**	–	–	7355↓	79.6
p_hat300-3	3774	**0.37**	2.24	1.39	3774	3.22
p_hat500-2	3920	**0.34**	2.17	0.64	3920	46.5
p_hat500-3	5375	**12.8**	803.6	113.2	5375	2265
p_hat700-2	5290	**1.04**	40.2	2.47	5290	12.1
p_hat700-3	7565	**29.6**	–	276.9	7565	102.6
san1000	1716	7.32	163.5	**1.58**	1716	10.6
san200_0.9_2	6082	**0.24**	1.29	4.30	6082	0.25
san200_0.9_3	4748	**2.53**	12.9	7.81	4748	12.6
san400_0.7_1	3941	**1.32**	2.99	2.97	3941	1.69
san400_0.7_2	3110	**3.75**	4.29	12.8	3110	**1.18**
san400_0.7_3	2771	**2.88**	5.64	9.72	2771	2126
san400_0.9_1	9776	70.9	1001	1893	9776	**0.99**
sanr200_0.9	5126	**1.71**	5.57	5.44	5126	1.98
sanr400_0.7	2992	25.4	24.0	74.4	2992	**12.5**

graphs that were not solved by any exact solvers within 5000s. Among the 34 instances, FastWClq did not find the optimal solution of three instances (marked with '↓'). In general, TSM-MWC greatly outperforms the compared solvers on the DIMACS graphs.

To evaluate TSM-MWC on massive graphs, we solved 215 real-world graphs from the Network Data Repository (Rossi and Ahmed 2015), including the 52 graphs used to evaluate WLMC in (Jiang, Li, and Manyà 2017), the 90 graphs used to evaluate FastWClq in (Cai and Lin 2016), and 25 hard graphs of brain networks. MWCLQ is not compared because it was not designed for massive graphs. The cutoff time was set to 1000s except for 3 biological graphs and the 25 graphs of brain networks, which used a cutoff time of 10000s. All the times, in seconds, include the preprocessing and search times, but not the time to read the input graphs.

Table 2 excludes the 168 graphs that were solved by both TSM-MWC and WLMC within 10s, and shows results for the remaining 47 graphs. The best times are in bold (FastWClq times are not in bold if the best solution found is not optimal). TSM-MWC solved the 47 instances of the table, and is faster than WLMC and FastWClq on 27 instances. WLMC did not solve 17 hard instances and FastWClq did not find the optimum of 29 instances (marked with '↓') within the cutoff time. Overall, TSM-MWC significantly outperforms WLMC

Table 2: Comparison of runtimes in seconds of TSM-MWC with WLMC and FastWClq on real-world massive graphs.

Instance *#cutoff=1000s*	$\omega_v(G)$	TSM-MWC	W-LWC	FastWClq best	FastWClq avgt
aff-digg	3836	**218.1**	756.0	2967↓	948.1
aff-orkut-user2groups	971	**279.0**	375.5	848↓	819.3
dbpedia-link	5062	**25.94**	26.67	4973↓	560.5
rec-dating	1699	**14.48**	15.23	1568↓	554.9
rec-libimseti-dir	1938	**11.30**	13.36	1938	468.5
rec-movielens	3777	**24.48**	35.80	3420↓	954.4
rgg_n_2_24_s0	2514	14.05	12.25	2514	**9.33**
scc_twitter-copen	58699	31.06	8.38	58699	**0.12**
sc-TSOPF-RS-b2383	960	**8.95**	43.28	960	230.9
socfb-konect	981	13.42	**11.07**	981	33.29
soc-flickr-und	10127	**49.52**	170.5	10126↓	514.9
soc-livejournal -user-groups	1054	**50.93**	59.62	991↓	608.8
soc-orkut-dir	6147	49.58	**47.93**	6147	64.36
soc-orkut	5452	45.42	**39.94**	5452	65.73
soc-sinaweibo	4759	**40.98**	52.31	4545↓	922.2
tech-p2p	18897	**748.1**	–	17250↓	871.8
twitter_mpi	13524	**246.7**	–	11801↓	639.6
web-wikipedia-growth	4741	**10.39**	11.39	4741	72.62
web-wikipedia_link_it	89947	140.4	**80.27**	2500↓	4.15
#cutoff=10000s for 28 hard instances of biological and brain networks					
bio-human-gene1	134713	6804	**2637**	134362↓	4571
bio-human-gene2	135310	5970	**1474**	135059↓	1097
bio-mouse-gene	59952	**1722**	4024	59855↓	1840
bn...864_session_1-bg	32294	**1764**	–	31496↓	7609
bn...864_session_2-bg	27190	1238	–	27190	**176.6**
bn...865_session_1-bg	29370	**1157**	1391	28544↓	7467
bn...865_session_2-bg	29870	**951.8**	–	29381↓	4688
bn...867_session_1-bg	29425	907.5	**676.2**	29208↓	5491
bn...867_session_2-bg	36021	1008	**711.9**	35428↓	7571
bn...868_session_1-bg	31940	1113	–	31940	**249.7**
bn...868_session_2-bg	29548	2403	–	29548	**107.7**
bn...869_session_1-bg	27957	**1121**	3075	27453↓	3555
bn...869_session_2-bg	29250	**1814**	–	29009↓	4823
bn...870_session_1-bg	28810	1047	–	28810	**126.0**
bn...870_session_2-bg	35415	**1329**	–	33944↓	2228
bn...871_session_1-bg	37828	1271	1357	37828	**383.5**
bn...871_session_2-bg	32835	1848	–	32835	**104.5**
bn...872_session_2-bg	35698	**1691**	–	35515↓	4000
bn...873_session_1-bg	29944	**5801**	–	29400↓	1537
bn...873_session_2-bg	32445	**765.3**	1277	32064↓	5330
bn...874_session_2-bg	30885	1309	1779	30885	**152.2**
bn...876_session_1-bg	50355	1282	–	50355	**584.6**
bn...876_session_2-bg	33085	**1506**	–	30829↓	5907
bn...878_session_1-bg	27775	675.7	4972	27775	**135.7**
bn...886_session_1	26281	**1211**	–	25548↓	610.4
bn...889_session_1	27500	**3153**	–	27003↓	5607
bn...889_session_2	24771	**807.0**	822.8	24497↓	7363
bn...912_session_2	35063	848.9	3110	35063	**33.1**

Table 3: The number of solved graphs (#) and the mean runtimes in seconds (avgt) of TSM-MWC, MWCLQ, WLMC and FastWClq for practical applications of MWC. The total number of graphs in each group is displayed between brackets in the first column.

Group	TSM-MWC		MWCLQ		WLMC		FastWClq	
	#	avgt	#	avgt	#	avgt	#	avgt
WDP (499)	**499**	**5.35**	**499**	10.1	**499**	14.6	350	48.2
ECC (15)	**15**	11.3	14	46.4	**15**	13.1	**15**	**3.82**
KES (100)	**82**	23.4	69	63.3	80	20.7	70	105
REF (129)	**106**	**2.21**	**106**	12.3	**106**	2.79	105	0.15

and FastWClq on the tested real-world massive graphs.

Table 3 compares the number of instances solved (i.e. an optimal solution was found) within a cutoff time of 3600s and the mean runtimes of TSM-MWC, MWCLQ, WLMC and FastWClq in the four groups of graphs coming from practical applications of MWC. TSM-MWC solves the most number of instances in every group, and is generally faster than the other solvers. For example, TSM-MWC solves 13, 2 and 12 KES graphs more than MWCLQ, WLMC and FastWClq, and is almost 2, 3 and 9 times faster than MWCLQ, WLMC and FastWClq on WDP graphs, respectively. Overall, TSM-MWC shows the best performance on the graphs coming from practical applications of MWC.

Table 4 compares the number of instances solved by TSM-MWC, MWCLQ and WLMC within the cutoff time of each group of instances, as well as the mean search tree size of the solved instances in each group. TSM-MWC solves the greatest number of instances in each group and its search trees are also the smallest, showing that the new two-stage MaxSAT reasoning approach implemented in TSM-MWC is more effective than the brute-force MaxSAT reasoning of MWCLQ and WLMC in reducing the search space. Note that the mean search tree size of TSM-MWC for the DIMACS graphs is greater than that of WLMC, because TSM-MWC solves six hard DIMACS graphs more than WLMC within the cutoff time and the mean search tree size is computed among the solved graphs.

Table 4: The number of solved graphs (#) and mean search tree size in 10^5 (tree) of TSM-MWC, MWCLQ and WLMC.

Group	TSM-MWC		MWCLQ		WLMC	
	#	tree	#	tree	#	tree
DIMACS (80)	**66**	60.6	61	144.9	60	15.5
MASSIVE (47)	**47**	**8.66**	-	-	30	16.6
WDP (499)	**499**	**1.92**	**499**	18.6	**499**	6.02
ECC (15)	**15**	**14.2**	14	168.3	**15**	20.5
KES (100)	**82**	**13.9**	69	351.1	80	27.8
REF (129)	**106**	**3.78**	**106**	70.4	**106**	4.22

Effects of the Two-stage MaxSAT Reasoning

To evaluate the individual effect of the two-stage MaxSAT reasoning (Algorithm 2 and Algorithm 3), we conducted an experiment with the following variants of TSM-MWC:

Table 5: The number of solved graphs (#) and the mean search tree size in 10^5 (tree) of B_{IS}, B_{Binary}, B_{MaxSAT} and $\mathrm{B}_{Ordered}$.

Group	B_{IS}		B_{Binary}		B_{MaxSAT}		$\mathrm{B}_{Ordered}$	
	#	tree	#	tree	#	tree	#	tree
DIMACS (80)	54	305	**64**	**198**	60	24.1	**66**	164
MASSIVE(47)	15	7.51	**34**	30.4	30	11.5	**35**	13.1
WDP (499)	**499**	74.2	**499**	**6.05**	**499**	8.69	**499**	**4.36**
ECC (15)	**15**	66.0	**15**	**25.0**	**15**	33.4	**15**	**26.1**
KES (100)	80	43.3	**82**	**21.4**	80	33.0	**81**	**30.2**
REF (129)	**106**	10.7	**106**	**5.89**	**106**	8.88	**106**	**8.48**

B_{Binary}: It is TSM-MWC, but the set B of branching vertices of Algorithm 1 is generated using only binary MaxSAT reasoning; i.e., the lines 3 and 4 of Algorithm 4 are removed.

B_{IS}: It is B_{Binary}, but the set B of branching vertices of Algorithm 1 is generated using the standard IS partition approach as in WLMC and MWCLQ: Let $v_1 < \cdots < v_n$ be an ordering over the vertices of the input graph and let the colors be represented by positive integers. For $i = n$ downto 1, it assigns the smallest possible color to v_i. An IS consists of the vertices with the same color.

$\mathrm{B}_{Ordered}$: It is B_{IS}, but the set B generated by the standard IS partition approach is further reduced using ordered MaxSAT reasoning (Algorithm 3). Unlike TSM-MWC, $\mathrm{B}_{Ordered}$ does not use binary MaxSAT reasoning.

B_{MaxSAT}: It is B_{IS}, but the set B generated by the standard IS partition approach is further reduced using brute-force MaxSAT reasoning as in WLMC, instead of using ordered MaxSAT reasoning as in TSM-MWC.

Table 5 compares the number of solved instances and the mean search tree size (in 10^5) of the four solvers using the graphs of Tables 1-3 and the same cutoff times. B_{Binary} solves more DIMACS, massive and KES graphs than B_{IS}, and generates smaller search trees, showing that binary MaxSAT reasoning generates smaller sets of branching vertices than the common IS partition approach. Similarly, $\mathrm{B}_{Ordered}$ solves more DIMACS, massive and KES graphs than B_{MaxSAT} and its search trees are generally smaller, showing that ordered MaxSAT reasoning is more efficient than brute-force MaxSAT reasoning in reducing the number of branching vertices. Note that TSM-MWC implements both binary and ordered MaxSAT reasoning and is substantially better than the four variants in Table 5.

Conclusions

We proposed TSM-MWC, a new exact algorithm for MWC that incorporates a novel two-stage MaxSAT reasoning approach to minimizing the number of branches: binary MaxSAT reasoning to generate an initial set of branching vertices and ordered MaxSAT reasoning to further reduce the number of branching vertices. The reported experiments show that the two-stage MaxSAT reasoning approach is very effective in reducing the search space, and that TSM-MWC outperforms relevant exact and heuristic MWC algorithms on small/medium graphs, real-world massive graphs and graphs from practical applications of MWC.

Acknowledgements

This work is supported by NSFC Grant No. 61272014, No. 61472147, No. 61370183 and No. 61370184, the Metrics platform of University of Picardie Jules Verne, the HPC platform of JHUN, and the MINECO-FEDER project RASO TIN2015-71799-C2-1-P.

References

Benlic, U., and Hao, J. 2013. Breakout local search for maximum clique problems. *Computers & Operations Research* 40(1):192–206.

Butenko, S., and Wilhelm, W. E. 2006. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research* 173(1):1–17.

Cai, S., and Lin, J. 2016. Fast solving maximum weight clique problem in massive graphs. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI, New York, NY, USA*, 568–574.

Fan, Y.; Li, N.; Li, C.; Ma, Z.; Latecki, L. J.; and Su, K. 2017. Restart and random walk in local search for maximum vertex weight cliques with evaluations in clustering aggregation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, 622–630.

Fang, Z.; Li, C. M.; and Xu, K. 2016. An exact algorithm based on MaxSAT reasoning for the maximum weight clique problem. *Journal of Artificial Intelligence Research* 55:799–833.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Jiang, H.; Li, C. M.; and Manyà, F. 2016. Combining efficient preprocessing and incremental MaxSAT reasoning for MaxClique in large graphs. In *Proceedings of 22nd European Conference On Artifical Intelligence, ECAI*, 939–947.

Jiang, H.; Li, C. M.; and Manyà, F. 2017. An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, California, USA*, 830–838.

Kumlander, D. 2008. On importance of a special sorting in the maximum-weight clique algorithm based on colour classes. In *Proceedings of Second International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences, MCO*, 165–174.

Li, C. M., and Quan, Z. 2010. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In *Proceedings of Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, 128–133.

Li, C. M.; Jiang, H.; and Manyà, F. 2017. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research* 84:1–15.

Mascia, F.; Cilia, E.; Brunato, M.; and Passerini, A. 2010. Predicting structural and functional sites in proteins by searching for maximum-weight cliques. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, 1274–1279.

McCreesh, C.; Prosser, P.; Simpson, K.; and Trimble, J. 2017. On maximum weight clique algorithms, and how they are evaluated. In *Principles and Practice of Constraint Programming, CP 2017*, 206–225.

Ostergard, P. 2002. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics* 120(1-3):197–207.

Pullan, W.; Mascia, F.; and Brunato, M. 2011. Cooperating local search for the maximum clique problem. *Journal of Heuristics* 17(2):181–199.

Rossi, R., and Ahmed, N. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI*, 4292–4293.

San, S.; Matia, F.; Rodriguez, L.; and Hernando, M. 2011. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research* 38(2):571–581.

Shimizu, S.; Yamaguchi, K.; Saitoh, T.; and Masuda, S. 2012. Some improvements on Kumlander's maximum weight clique extraction algorithm. In *Proceedings of the International Conference on Electrical, Computer, Electronics and Communication Engineering*, 307–311.

Shimizu, S.; Yamaguchi, K.; Saitoh, T.; and Masuda, S. 2013. Optimal table method for finding the maximum weight clique. In *Proceedings of the 13th International Conference on Applied Computer Science, ACS*, 84–90.

Tomita, E.; Sutani, Y.; Higashi, T.; Takahashi, S.; and Wakatsuki, M. 2010. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *WALCOM: Algorithms and Computation, 4th International Workshop, WALCOM 2010*, 191–203.

Wang, Y.; Cai, S.; and Yin, M. 2016. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI*, 805–811.

Wu, Q., and Hao, J. 2013. An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization* 26(1):86–108.

Wu, Q., and Hao, J. 2015a. A review on algorithms for maximum clique problems. *European Journal of Operational Research* 242(3):693–709.

Wu, Q., and Hao, J. 2015b. Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications* 42(1):355–365.

Wu, Q.; Hao, J.; and Glover, F. 2012. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research* 196(1):611–634.

Zhang, D.; Javed, O.; and Shah, M. 2014. Video object co-segmentation by regulated maximum weight cliques. In *Proceedings of the13th European Conference on Computer Vision, ECCV, Zurich, Switzerland*, 551–566.

Zhian, H.; Sabaei, M.; Javan, N. T.; and Tavallaie, O. 2013. Increasing coding opportunities using maximum-weight clique. In *Proceedings of The 5th Computer Science and Electronic Engineering Conference, CEEC*, 168–173.

Zhou, Y.; Hao, J.; and Goëffon, A. 2017. PUSH: A generalized operator for the maximum vertex weight clique problem. *European Journal of Operational Research* 257(1):41–54.