Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

An Exact Algorithm for the Maximum Weight Clique Problem in Large Graphs

Hua Jiang
Huazhong University
of Science and Technology, China
email: jh_hgt@163.com

Chu-Min Li*
MIS, Université de Picardie
Jules Verne, France
email: chu-min.li@u-picardie.fr

Felip Manyà
Artificial Intelligence Research
Institute (IIIA), CSIC, Spain
email: felip@iiia.csic.es

Abstract

We describe an exact branch-and-bound algorithm for the maximum weight clique problem (MWC), called WLMC, that is especially suited for large vertex-weighted graphs. WLMC incorporates two original contributions: a preprocessing to derive an initial vertex ordering and to reduce the size of the graph, and incremental vertex-weight splitting to reduce the number of branches in the search space. Experiments on representative large graphs from real-world applications show that WLMC greatly outperforms relevant exact and heuristic MWC algorithms, and refute the prevailing hypothesis that exact MWC algorithms are less adequate for large graphs than heuristic algorithms.

Introduction

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

MWC is *NP-Hard* and has practical applications in different domains such as protein structure prediction (Mascia et al. 2010), coding theory (Zhian et al. 2013), combinatorial auctions (Wu and Hao 2015; Fang, Li, and Xu 2016) and computer vision (Ma and Latecki 2012; Zhang, Javed, and Shah 2014).

The main objective of this paper is to develop an exact and highly competitive MWC algorithm for large graphs. The focus on large graphs is motivated by the fact that they are ubiquitous: computer networks, social networks, mobile call networks, biological networks, citation networks, and the World Wide Web, to name a few. These networks typically have very low density, a huge number of vertices, and common statistical properties such as small-world property, power-law degree distributions, and clustering (Newman 2003). Finding cliques is very relevant in this context. For example, a clique might be a functional group in biological networks, and identify a community in social networks.

There exist a few exact MC algorithms for large graphs. The best performing ones are PMC (Rossi et al. 2013), BBMCSP (San Segundo, Alvaro, and Pardalos 2016) and LMC (Jiang, Li, and Manyà 2016), based on the branch-and-bound (BnB) scheme. Nevertheless, their graph preprocessing, upper bound (UB) computation and branching strategy are not suitable for MWC. Very few exact and heuristic MWC algorithms have been proposed, compared with the number of available MC algorithms. This is partially due to the fact that MWC is more complicated than MC and some successful MC techniques are not applicable or ineffective for MWC because of the vertex weights (Cai and Lin 2016).

To our knowledge, the two most efficient heuristic MWC algorithms for large graphs are based on local search: LSCC+BMS (Wang, Cai, and Yin 2016), and FastWClq (Cai and Lin 2016). Most exact MWC algorithms implement the BnB scheme and differ in their UB computation and branching strategy. The most relevant ones are Cliquer (Ostergard 2001; 2002), Kumlander's algorithm (Kumlander 2004; 2008), VCTable (Shimizu et al. 2012), OTClique (Shimizu et al. 2013), and MWCLQ (Fang, Li, and Xu 2016). MWC can also be solved exactly via its reduction to MinSAT (Li et al. 2012).

Unfortunately, all exact MWC algorithms in our knowledge exhibit poor performance on large graphs, and because of this we have developed an exact BnB MWC algorithm for large graphs, called WLMC (short for Weighted Large Maximum Clique), that incorporates two important contributions of the paper: a preprocessing to derive an initial vertex ordering and to reduce the size of the graph by removing vertices not belonging to any optimal solution, and incremental vertex-weight splitting to reduce the number of branches in the search space.

We have also conducted experiments using real-world graphs that show that WLMC greatly outperforms relevant exact and heuristic algorithms on large graphs. This is another important contribution of the paper: the performance of WLMC refutes the prevailing hypothesis that exact MWC algorithms, despite proving optimality, are less adequate for

*Corresponding author

large graphs than heuristic algorithms.

The paper is organized as follows: Section 2 describes WLMC and the techniques it implements. Section 3 analyzes the empirical results. Section 4 gives the conclusions.

WLMC: A New Exact MWC Algorithm

WLMC contains two main components: an efficient preprocessing procedure *Initialize* to derive an initial vertex ordering and to reduce the size of the graph, and a BnB algorithm *SearchMaxWClique* that implements incremental vertex-weight splitting to reduce the number of branches in the search space. We first describe the two components and then algorithm WLMC.

The Efficient Preprocessing Procedure *Initialize*

Initialize has as input a graph $G = (V, E, w)$ and a lower bound lb of $\omega_v(G)$. It returns an initial vertex ordering O_0, an initial clique C_0, and a reduced graph G' of G. The pseudo-code of *Initialize* is shown in Algorithm 1.

Initialize works on a copy $H = (U, E, w)$ of G to compute an initial vertex ordering O_0: $v_1 < v_2 < \cdots < v_{|V|}$ of G, in which v_1 is the vertex with the smallest degree in H, v_2 is the vertex with the smallest degree in H after v_1 is removed, and so on. This ordering is used for MC in (Carraghan and Pardalos 1990) and is showed to be also effective for solving MC on large graphs in (Jiang, Li, and Manyà 2016). The intuition behind the ordering is that the greater vertices have more chance to form larger cliques. After removing $v_1, v_2, \ldots, v_{i-1}$ from H, if the vertex v_i with the smallest degree $deg(v_i)$ is adjacent to all the other vertices in H (line 6), H becomes a clique C_0, because the degree of other vertices of H is necessarily equal to $deg(v_i)$ in this case. The vertices in C_0 can be ordered arbitrarily (line 8). After C_0 is obtained, lb is updated to $w(C_0)$ if $w(C_0) > lb$, and all the vertices v such that the total weight of v and its neighbors in G, say $w^*(v)$, is not greater than lb are removed from G to derive the reduced graph G', because they cannot belong to any clique of weight greater than lb. Finally, *Initialize* returns C_0, O_0 and G'.

The number of neighbors of a vertex is in $O(|V|)$. Computing the degree of all the vertices can be done in $O(|V|^2)$ or $O(|E|)$ time ($|E|$ is in $O(|V|^2)$ in the worst case). Searching for the vertex with the smallest degree needs $O(|V|)$ time. So, the time complexity of $Initialize$ is $O(|V|^2)$ in the worst case. Note that *Initialize* does not use the notion of k-$core$, which is effective for solving MC on large graphs in (Jiang, Li, and Manyà 2016), but is ineffective for MWC.

The Procedure *SearchMaxWClique*

Given a graph $G = (V, E, w)$, the best clique C_{max} so far and an ordering O over V, the BnB procedure *SearchMaxWClique* searches recursively for a clique of weight greater than $w(C_{max})$, combined with the growing clique C. In the sequel, $\Gamma(v)$ denotes the set of vertices that are adjacent to v, $G[P]$ denotes the subgraph of G induced by the subset of vertices P ($P \subseteq V$), and $w_{max}(P)$ ($w(P)$) denotes the biggest (total) weight of the vertices in P.

Algorithm 1: Initialize(G, lb)

Input: $G = (V, E, w)$, a lower bound lb of $\omega_v(G)$
Output: an initial clique C_0, an initial vertex ordering O_0, and a reduced graph G' of G

```
1  begin
2      U ← V;
3      compute the degree deg(v) for each vertex v in U;
4      for i:= 1 to |V| do
5          v_i ← the vertex v with the smallest deg(v) in U;
6          if deg(v_i) = |U| − 1 then
7              /* v_i is adjacent to all other vertices in U */
8              order U arbitrarily as {v_i, v_{i+1}, ..., v_{|V|}};
9              C_0 ← U; break;
10         U ← U \ {v_i};
11         for each neighbor v of v_i in U do
12             deg(v) ← deg(v) − 1;
13     if w(C_0) > lb then lb ← w(C_0);
14     let w*(v) denote the total weight of v and its
         neighbors in G;
15     G' ← G after removing vertices v s.t. w*(v) ≤ lb;
16     O_0 ← v_1 < v_2 < ··· < v_{|V|};
17     return (C_0, O_0, G');
```

Algorithm 2 shows the pseudo-code of *SearchMaxWClique*. If the set of vertices V is non-empty, it calls function *GetBranches* to partition V into two sets A and B in such a way that the maximum weight of a clique in $G[A]$ is not greater than $w(C_{max}) - w(C)$, and $B = \{b_1, b_2, \ldots, b_{|B|}\}$ is the returned set of branching vertices. If B is empty, the search is pruned and the current best clique C_{max} is returned. Otherwise, it recursively searches for a maximum weight clique in $G[\Gamma(b_i) \cap (\{b_{i+1}, b_{i+2}, \ldots, b_{|B|}\} \cup A)]$, to be added in $C \cup \{b_i\}$, for $i = |B|, \ldots, 1$. Note that the algorithm iterates over B in the inverse ordering of O, because O is computed by the procedure *Initialize* and greater vertices w.r.t. O have more chance to form larger cliques.

Algorithm 3 describes function *GetBranches*(G, t, O), where t is an integer representing a weight and O is an ordering over the vertices of G. *GetBranches* works in two phases. In the first phase (lines 2–11), it computes a set of independent sets (ISs) $\Pi = \{D_1, D_2, \ldots, D_{|\Pi|}\}$ by sequentially inserting vertices of G, starting from the greatest w.r.t. O, into these ISs, provided that $\sum_{j=1}^{|\Pi|} w_{max}(D_j) \leq t$. Let $A = V(\Pi) = D_1 \cup \cdots \cup D_{|\Pi|}$ be the set of vertices occurring in Π. Observe that any clique C_A of $G[V(\Pi)]$ contains at most one vertex from each IS D_i, $1 \leq i \leq |\Pi|$. So, $w(C_A) \leq \sum_{j=1}^{|\Pi|} w_{max}(D_j) \leq t$.

The vertices of G that cannot be inserted into any IS, because then $\sum_{j=1}^{|\Pi|} w_{max}(D_j) > t$, form the set of branching vertices B. As a result, we have B and an IS partition $\Pi = \{D_1, D_2, \ldots, D_{|\Pi|}\}$ of $A = V \setminus B$.

Example 1. Let $G = (V, E, w)$ be the graph of Figure 1, where $v_i^{w_i}$ denotes vertex v_i with weight $w_i = w(v_i)$, and let O: $v_1 < v_2 < \cdots < v_6$ be the vertex ordering. As-

Algorithm 2: SearchMaxWClique(G, C_{max}, C, O)

Input: $G = (V, E, w)$, the best clique C_{max} so far, the current growing clique C, a vertex ordering O
Output: Clique C if $w(C)>w(C_{max})$; otherwise C_{max}

```
1  begin
2      if |V| = 0 then return C;
3      B ← GetBranches(G, w(C_max) − w(C), O);
4      if B = ∅ then return C_max;
5      A ← V \ B;
6      Let B={b_1, b_2, ..., b_|B|}, b_1<b_2<···<b_|B| w.r.t. O;
7      for i := |B| to 1 do
8          P ← Γ(b_i) ∩ ({b_{i+1}, b_{i+2}, ..., b_|B|} ∪ A);
9          if w(C ∪ {b_i}) + w(P) > w(C_max) then
10             C' ← SearchMaxWClique(G[P], C_max,
11                                   C∪{b_i}, O);
12             if w(C') > w(C_max) then C_max ← C';
13     return C_max;
```

Algorithm 3: GetBranches(G, t, O)

Input: $G = (V, E, w)$, an integer t and an ordering O
Output: a set B of branching vertices

```
1  begin
2      B ← ∅; Π ← ∅; /* Π will be a set of ISs*/
3      while V is non-empty do
4          v ← the greatest vertex of V w.r.t. O;
5          V ← V \ {v};
6          if ∃D ∈ Π s.t. Γ(v) ∩ D = ∅ and
7            Σ_{j=1}^{|Π|} w_max(D_j) ≤ t after adding v into D
8          then D ← D ∪ {v};
9          else if Σ_{j=1}^{|Π|} w_max(D_j) + w(v) ≤ t then
10             create a new IS D = {v}, Π ← Π ∪ {D};
11         else B ← B ∪ {v};
12     ub_0 ← Σ_{j=1}^{|Π|} w_max(D_j);
13     Let B={b_1, b_2, ..., b_|B|}, b_1<b_2<···<b_|B| w.r.t. O;
14     for i := |B| to 1 do
15         (ub, Π') ← UP&Split(G, Π ∪ {{b_i}},
16                             ub_0 + w(b_i), t);
17         if ub ≤ t then
18             ub_0 ← ub, Π ← Π', B ← B \ {b_i};
19     return B;
```

sume that the best clique weight so far is $w(C_{max}) = 6$, and we call GetBranches(G, 6, O). During the first phase, GetBranches inserts the vertices $v_6^1, v_5^4, v_4^1, v_3^2, v_2^3$ into two ISs: $D_1 = \{v_6^1, v_5^4, v_2^3\}$ and $D_2 = \{v_4^1, v_3^2\}$. Afterwards, v_1^3 has adjacent vertices in both D_1 and D_2, and GetBranches cannot create a new IS $D_3 = \{v_1^3\}$ because then $\sum_{j=1}^{3} w_{max}(D_j) > 6$. Hence, the first phase gives $A = \{v_6^1, v_5^4, v_4^1, v_3^2, v_2^3\}$ and $B = \{v_1^3\}$.

The second phase of *GetBranches* (lines 12–18) tries to remove each vertex $b_i \in B$ from B and insert it into A.

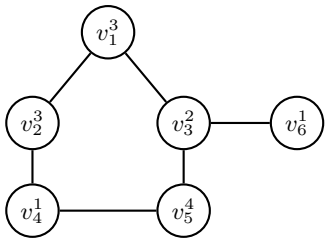

Figure 1: A graph with $\omega_v(G)$=6

Recall that $\Pi = \{D_1, D_2, \ldots, D_{|\Pi|}\}$ and $A = V(\Pi)$. In order to insert b_i into A, we have to show that $G[A \cup \{b_i\}]$ does not contain any clique of weight greater than t. Since any clique in $G[A \cup \{b_i\}]$ is formed by at most one vertex from each IS of Π and possibly by b_i, an upper bound of its weight is $\sum_{j=1}^{|\Pi|} w_{max}(D_j) + w(b_i)$. This upper bound is very conservative because it is tight only if the clique is formed by the most weighted vertex of every IS of Π and by b_i. However, a set of q ISs often cannot form a clique containing q vertices, and such a set is said to be *conflicting* (Li and Quan 2010). The main task of the second phase of *GetBranches* is to improve this upper bound by identifying as many conflicting subsets of ISs as possible, inspired by the MaxSAT reasoning in (Li and Quan 2010; Fang, Li, and Xu 2016). If the improved upper bound is not greater than t, b_i is removed from B and is added to A.

Example 2. Let us illustrate how to identify a conflicting subset of ISs to improve an upper bound by continuing with the graph G of Figure 1. In the first phase, *GetBranches* gives $\Pi = \{D_1, D_2\}$, $D_1 = \{v_6^1, v_5^4, v_2^3\}$, $D_2 = \{v_4^1, v_3^2\}$, $B = \{v_1^3\}$ and $t = 6$ (cf. Example 1). Our objective here is to show that the vertices in $\Pi \cup \{\{v_1^3\}\}$ cannot form any clique of weight greater than 6.

The initial upper bound for $\Pi \cup \{\{v_1^3\}\}$ is $w_{max}(D_1) + w_{max}(D_2) + w(v_1^3) = 9$. If v_1^3 is in a clique C_G of G, then v_4^1, v_5^4 and v_6^1 cannot be in C_G because they are not adjacent to v_1^3. So, we remove v_4^1 from D_2, and v_5^4 and v_6^1 from D_1. Since D_1 becomes unit, its unique vertex v_2^3 is added to C_G, which removes v_3^2 from D_2 because v_3^2 is not adjacent to v_2^3, and D_2 becomes empty. This reasoning shows that if v_1^3 is in a clique C_G and D_1 contains a vertex in C_G, then D_2 cannot contain any vertex in C_G. So, D_1, D_2 and D_3 are conflicting, because one of these ISs cannot contain any vertex in C_G. Since $min(w_{max}(D_1), w_{max}(D_2), w_{max}(D_3)) = 2$, the initial upper bound is improved from 9 to $9 - 2 = 7$.

Observe that the improved upper bound is tight only if C_G contains the most weighted vertices v_5^4 and v_1^3, which is impossible because v_5^4 and v_1^3 are not adjacent in G. A further improvement of the upper bound can be obtained by splitting all the weights greater than 2 in D_1, D_2 and $\{v_1^3\}$. The splitting gives a set of ISs $\Pi_1 = \{\{v_6^1, v_5^2, v_2^2\}, \{v_4^1, v_3^2\}, \{v_1^2\}\}$, in which the maximum weight in each IS is 2, and a set of ISs $\Pi_R = \{\{v_5^2, v_2^1\}, \{v_1^1\}\}$ consisting of the weights split from Π. Note that the splitting of Π is equivalent to splitting G into the two graphs G_1 and G_R of Figure 2, where

the vertices with weight 0 in G_R are added to facilitate the understanding. Any clique C_G in G is also a clique C_{G_1} in G_1 and a clique C_{G_R} in G_R. It holds that $w(C_G) = w(C_{G_1}) + w(C_{G_R})$, because the weight of any vertex v in G is equal to the sum of weights of v in G_1 and G_R. Let UB_{G_1} (UB_{G_R}) denote an upper bound of the weight of C_{G_1} (C_{G_R}). It holds that $w(C_G) \leq UB_{G_1} + UB_{G_R}$. In other words, $UB_{G_1} + UB_{G_R}$ is an upper bound of $\omega_v(G)$.

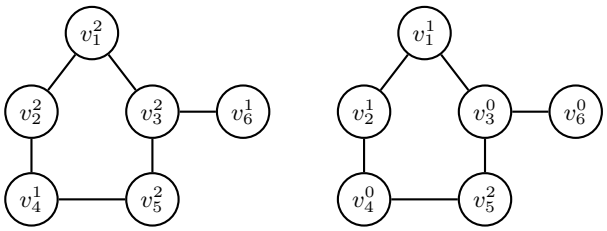

Figure 2: Graphs G_1 (left) and G_R (right) obtained by splitting the graph G of Figure 1

Since Π_1 partitions G_1 and is conflicting, for any clique in G_1, there is an IS of Π_1 such that the clique does not contain any vertex from the IS. So, $UB_{G_1} = 2 + 2 + 2 - 2 = 4$.

$\Pi_R = \{\{v_1^1\}, \{v_5^2, v_2^1\}\}$ partitions G_R after removing all the vertices with weight 0. If v_1^1 is in a clique, then the most weighted vertex v_5^2 in $\{v_5^2, v_2^1\}$ cannot be in the clique, because v_1^1 and v_5^2 are not adjacent. Consequently, $\{v_5^2, v_2^1\}$ can be split into $\{v_5^1\}$ and $\{v_5^1, v_2^1\}$, so that $\{v_1^1\}$ and $\{v_5^1\}$ are conflicting, suggesting us to split G_R into the graphs G_2 and G_3 of Figure 3, where G_2 can be partitioned into $\Pi_2 = \{\{v_1^1\}, \{v_5^1\}\}$ and G_3 can be partitioned into $\Pi_3 = \{\{v_5^1, v_2^1\}\}$, after removing all the vertices with weight 0.

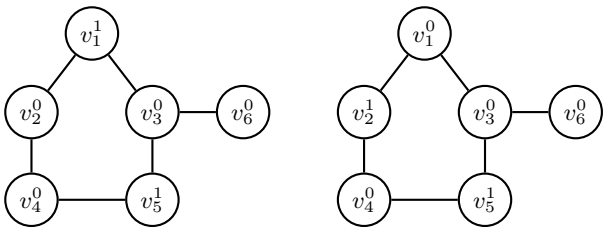

Figure 3: Graphs G_2 (left) and G_3 (right) obtained by splitting the graph G_R of Figure 2

Any clique C_{G_R} in G_R is also a clique C_{G_2} in G_2 and a clique C_{G_3} in G_3. It holds that $w(C_{G_R}) = w(C_{G_2}) + w(C_{G_3})$. Let UB_{G_2} (UB_{G_3}) denote an upper bound of $\omega_v(G_2)$ ($\omega_v(G_3)$). Then $UB_{G_2} + UB_{G_3}$ is an upper bound of $\omega_v(G_R)$. So, $w(C_G) \leq UB_{G_1} + UB_{G_2} + UB_{G_3}$. Clearly, $UB_{G_2} = UB_{G_3} = 1$. So $w(C_G) \leq 4 + 1 + 1 = 6$, meaning that v_1^3 can be removed from B and added to A.

An IS containing exactly one vertex is a *unit* IS, and so $\Pi \cup \{\{b_i\}\}$ contains at least one unit IS. Example 2 illustrates how to propagate the unit IS $\{b_i\}$: repeatedly select a unit IS $\{v\}$ and remove all the vertices non-adjacent to v from the other ISs, possibly resulting in new unit ISs, until an empty IS is produced or there is no more unit IS. If an empty IS S_0 is produced, we retrace the unit IS propagation to identify all ISs responsible to produce the empty IS, obtaining a conflicting subset of ISs $\{S_0, S_1, S_2, \ldots, S_r\}$. Let $\delta = min(w_{max}(S_0), \ldots, w_{max}(S_r))$, we split each weight greater than δ in S_j, $0 \leq j \leq r$, to obtain S'_j and S''_j so that $w_{max}(S'_j) = \delta$ and $w_{max}(S''_j) = w_{max}(S_j) - \delta$. For instance, in Example 2, with $\delta = 2$, the IS $\{v_6^1, v_5^4, v_2^3\}$ is split into $S' = \{v_6^1, v_5^2, v_2^2\}$ and $S'' = \{v_5^2, v_2^1\}$ by splitting all the weights w greater than δ into δ and $w - \delta$ and by keeping all the weights smaller than or equal to δ in S'. Consequently, we obtain a conflicting subset of ISs $\{S'_0, S'_1, S'_2, \ldots, S'_r\}$, in which $w_{max}(S'_j) = \delta$ for each j, $0 \leq j \leq r$, and a subset of ISs $\{S''_0, S''_1, S''_2, \ldots, S''_r\}$ from which further conflicts can be detected. The set of conflicting ISs $\{S'_0, S'_1, S'_2, \ldots, S'_r\}$ allows to improve the upper bound by δ, because at least one IS cannot contribute any of its vertices to form a clique.

In some cases, unit IS propagation does not result in an empty IS, but removes the most weighted vertices from an IS. A set of conflicting ISs can also be identified in these cases. For example, let $S_0 = \{v_1^7, v_2^4, v_3^3, v_4^1\}$. Assume that unit IS propagation involving the ISs S_1, S_2 and S_3 removes v_1^7 and v_2^4 from S_0, and we have $min(w_{max}(S_1), w_{max}(S_2), w_{max}(S_3)) = 2$. We split S_0 into $S'_0 = \{v_1^2, v_2^1\}$ and $S''_0 = \{v_1^5, v_2^3, v_3^3, v_4^1\}$, and S_j into S'_j and S''_j so that $w_{max}(S'_j) = 2$ and $w_{max}(S''_j) = w_{max}(S_j) - 2$ for each j, where $1 \leq j \leq 3$. Clearly, $\{S'_0, S'_1, S'_2, S'_3\}$ is a set of conflicting ISs in which $w_{max}(S'_j) = 2$ for each j, where $0 \leq j \leq 3$. The weight of v_1 and v_2 in S'_0 and S''_0 is determined as follows. Their weight in S''_0 should be at least 3 so that v_1 and v_2 remain to be the most weighted vertices in S''_0 to ensure $w_{max}(S_0) = w_{max}(S'_0) + w_{max}(S''_0)$. The weight of v_1 in S'_0 is then $min(7 - 3, w_{max}(S_1), w_{max}(S_2), w_{max}(S_3)) = 2$, and the weight of v_2 in S'_0 is then $min(4 - 3, w_{max}(S_1), w_{max}(S_2), w_{max}(S_3)) = 1$.

Generally speaking, let $S_0 = \{u_1^{w_1}, \ldots, u_k^{w_k}, \ldots, u_{|S_0|}^{w_{|S_0|}}\}$ be an IS in which the k most weighted vertices $u_1^{w_1}, \ldots, u_k^{w_k}$ are removed by unit IS propagation involving the ISs $S_1, S_2, \ldots, S_r$. Without loss of generality, assume that $w_1 \geq \cdots \geq w_k \geq w_{k+1} \geq \cdots \geq w_{|S_0|}$. Let $\delta = min(w_1 - w_{k+1}, w_{max}(S_1), \ldots, w_{max}(S_r))$ and let $w''_j = w_j - min(\delta, w_j - w_{k+1})$ for $1 \leq j \leq k$. We split S_0 into $S'_0 = \{u_1^{min(\delta, w_1 - w_{k+1})}, \ldots, u_k^{min(\delta, w_k - w_{k+1})}\}$ and $S''_0 = \{u_1^{w''_1}, \ldots, u_k^{w''_k}, u_{k+1}^{w_{k+1}}, \ldots, u_{|S_0|}^{w_{|S_0|}}\}$, and S_j into S'_j and S''_j so that $w_{max}(S'_j) = \delta$ and $w_{max}(S''_j) = w_{max}(S_j) - \delta$ for each j, where $1 \leq j \leq r$.

It holds that: (1) $min(\delta, w_1 - w_{k+1}) \geq min(\delta, w_2 - w_{k+1}) \geq \cdots \geq min(\delta, w_k - w_{k+1})$ in S'_0; (2) $w''_1 \geq w''_2 \geq \cdots \geq w''_k \geq w_{k+1} \geq \cdots \geq w_{|S_0|}$ in S''_0. To see (2), note that, for any numbers x_1, x_2 and x_3, we have $x_1 + min(x_2, x_3) = min(x_1 + x_2, x_1 + x_3)$, and $x_1 - min(x_2, x_1) \geq 0$. So, for $1 \leq j < k$, $w''_j - w''_{j+1} = w_j - min(\delta, w_j - w_{k+1}) - w_{j+1} + min(\delta, w_{j+1} - w_{k+1}) =$

$w_j - w_{j+1} + min(\delta, w_{j+1} - w_{k+1}) - min(\delta, w_j - w_{k+1}) = min(w_j - w_{j+1} + \delta, w_j - w_{k+1}) - min(\delta, w_j - w_{k+1}) \geq 0$; furthermore, $w''_k - w_{k+1} = w_k - min(\delta, w_k - w_{k+1}) - w_{k+1} = w_k - w_{k+1} - min(\delta, w_k - w_{k+1}) \geq 0$.

From (1) and (2), we easily see that $w_{max}(S'_0) + w_{max}(S''_0) = min(\delta, w_1 - w_{k+1}) + w''_1 = min(\delta, w_1 - w_{k+1}) + w_1 - min(\delta, w_1 - w_{k+1}) = w_1 = w_{max}(S_0)$.

Therefore, if unit IS propagation involving the ISs $S_1, S_2, \ldots, S_r$ removes the most weighted vertices of an IS S_0, we can split the weights to obtain a conflicting subset of ISs $\{S'_0, S'_1, S'_2, \ldots, S'_r\}$, and a subset of ISs $\{S''_0, S'_1, S'_2, \ldots, S'_r\}$ from which further conflicts can be detected.

Algorithm 4 implements the function *UP&Split* which performs unit IS propagation in a set of ISs $\Pi \cup \{\{b_i\}\}$. Every time an empty IS is produced or the most weighted vertices of an IS are removed, it calls the function $split(S, \delta)$ to split each involved IS S into two ISs S' and S'' as follows. Let $S = \{u_1^{w_1}, \ldots, u_{|S|}^{w_{|S|}}\}$ with $w_1 \geq \cdots \geq w_k \geq \delta \geq w_{k+1} \geq \cdots \geq w_{|S|}$, $split(S, \delta)$ returns $S' = \{u_1^{\delta}, \ldots, u_k^{\delta}, u_{k+1}^{w_{k+1}}, \ldots, u_{|S|}^{w_{|S|}}\}$ and $S'' = \{u_1^{w_1 - \delta}, \ldots, u_k^{w_k - \delta}\}$. In other words, each weight w_i greater than δ is split into δ and $w_i - \delta$, and each weight w_i not greater than δ remains in S'. Observe that $w_{max}(S) = w_{max}(S') + w_{max}(S'')$. As a result, Algorithm 4 obtains a subset of conflicting ISs and continues applying unit IS propagation to other ISs. Note that the sum of the maximum weights of the ISs is not changed by the splittings.

Finally, Algorithm 4 transforms $\Pi \cup \{\{b_i\}\}$ into $\Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_p$, where each $\Pi_j = \{S_{j1}, S_{j2}, \ldots, S_{j|\Pi_j|}\}$, $1 \leq j \leq p$, is a subset of ISs formed by some vertices occurring in $\Pi \cup \{\{b_i\}\}$, and each Π_j has an associated weight function w^j. The transformation fulfills the following conditions:

1. For each Π_j, $1 \leq j \leq p$, $S_{jk} \cap S_{jk'} = \emptyset$ if $k \neq k'$.
2. For each vertex $v \in V(\Pi \cup \{\{b_i\}\})$, the set of vertices occurring in $\Pi \cup \{\{b_i\}\}$, $w(v) = \sum_{j=1}^{p} w^j(v)$, where w is the weight function of G, w^j is the weight function associated with Π_j, and $w^j(v) = 0$ if $v \notin V(\Pi_j)$.
3. $\sum_{j=1}^{|\Pi|} w_{max}(D_j) + w(b_i) = \sum_{j=1}^{p} \sum_{k=1}^{|\Pi_j|} w^j_{max}(S_{jk})$.
4. For each j, $1 \leq j < p$, Π_j is a set of conflicting ISs, and $w^j_{max}(S_{j1}) = w^j_{max}(S_{j2}) = \cdots = w^j_{max}(S_{j|\Pi_j|})$.

Let C be a maximum weight clique of $G[V(\Pi \cup \{\{b_i\}\})]$, and let UB_j be an upper bound of the weight of C using the weight function w^j. Since $w(C) = \sum_{j=1}^{p} w^j(C)$ (Condition 2) and $w^j(C) \leq UB_j$, it holds that $w(C) \leq \sum_{j=1}^{p} UB_j$. By Condition 4, $UB_j \leq (|\Pi_j| - 1) \times w^j_{max}(S_{j1})$ for each $j < p$. Hence, $w(C) \leq \sum_{j=1}^{p-1}(|\Pi_j| - 1) \times w^j_{max}(S_{j1}) + \sum_{k=1}^{|\Pi_p|} w^p_{max}(S_{pk}) = \sum_{j=1}^{|\Pi|} w_{max}(D_j) + w(b_i) - \sum_{j=1}^{p-1} w^j_{max}(S_{j1})$ by Condition 3, which is the improved upper bound ub of $w(C)$ returned by UP&Split together with $\Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_p$.

If $ub \leq t$, b_i is removed from B and is added to A. Then, the reasoning on b_{i-1} is performed using $\Pi_p \cup \{\{b_{i-1}\}\}$.

Algorithm 4: UP&Split(G, Π, ub, t)

Input: $G = (V, E, w)$, $\Pi = \{D_1, D_2, \ldots, D_{|\Pi|}\}$ is a set of ISs, ub and t are positive integer
Output: the improved ub and transformed Π

```
begin
    Δ ← ∅;
    while there is a non-marked unit IS {v} in Π do
        remove vertices non-adjacent to v from their IS;
        if there is a non-marked empty IS S_0 then
            restore all the removed vertices into their IS;
            Let S_1, S_2, ..., S_r be the ISs responsible of
              removing all the vertices of S_0;
            δ ← min(w_max(S_0), ..., w_max(S_r));
            for each IS S_j in {S_0, S_1, S_2, ..., S_r} do
                (S'_j, S''_j) ← split(S_j, δ);
            Δ ← Δ ∪ {S'_0, S'_1, ..., S'_r};
            Π ← (Π \ {S_0, ..., S_r}) ∪ {S''_0, ..., S''_r};
            ub ← ub − δ;
        else if there is a non-marked IS S_0 in which the
          k most weighted vertices are removed then
            Let S_0 = {u_1^{w_1}, ..., u_k^{w_k}, ..., u_{|S_0|}^{w_{|S_0|}}} with
              w_1 ≥ ··· ≥ w_k ≥ w_{k+1} ≥ ··· ≥ w_{|S_0|};
            Let u_1^{w_1}, ..., u_k^{w_k} be the k most weighted
              vertices removed from S_0;
            β ← w_1 − w_{k+1};
            restore all the removed vertices into their IS;
            Let S_1, S_2, ..., S_r be the ISs responsible of
              removing u_1^{w_1}, ..., u_k^{w_k} from S_0;
            δ ← min(β, w_max(S_1), ..., w_max(S_r));
            γ ← w_{k+1};
            S'_0 ← {u_1^{min(δ,w_1−γ)}, ..., u_k^{min(δ,w_k−γ)}};
            Let w''_j be w_j − min(δ, w_j − γ) (1≤j≤k);
            S''_0 ← {u_1^{w''_1}, ..., u_k^{w''_k}, u_{k+1}^{w_{k+1}}, ..., u_{|S_0|}^{w_{|S_0|}}};
            for each IS S_j in {S_1, S_2, ..., S_r} do
                (S'_j, S''_j) ← split(S_j, δ);
            Δ ← Δ ∪ {S'_0, S'_1, ..., S'_r};
            Π ← (Π \ {S_0, ..., S_r}) ∪ {S''_0, ..., S''_r};
            ub ← ub − δ;
        if ub ≤ t then
            mark all ISs in Δ; break;
    restore all the removed vertices into their IS;
    return (ub, Δ ∪ Π);
```

Note that $\Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_{p-1}$ is not used for reasoning on b_{i-1} in this case, because this set is formed by all the conflicting subsets of ISs used to remove b_i and are marked as such. If $ub > t$, b_i is not removed from B, Π is not transformed, and $\Pi \cup \{\{b_{i-1}\}\}$ is used for reasoning on b_{i-1}.

GetBranches performs IS splitting incrementally. It first performs IS splitting in $\Pi \cup \{\{b_{|B|}\}\}$, obtaining an improved upper bound ub and a set Π' of ISs; If $ub \leq t$, it performs IS splitting in $\Pi' \cup \{\{b_{|B|-1}\}\}$, and so on. The approach here

is different from the BnB MWC algorithm MWCLQ (Fang, Li, and Xu 2016). MWCLQ, at every search tree node, encodes $G = (V, E, w)$ to a so-called LW-MaxSAT instance ϕ and performs MaxSAT reasoning to split the soft clauses in ϕ using two sophisticated inference rules called δ-$rule$ and (k, δ)-$rule$, to compute an upper bound of $w_v(G)$. MaxSAT reasoning in MWCLQ is not incremental and is not used for reducing the set of branching vertices, because it encodes the whole G into ϕ and considers simultaneously all the clauses of ϕ. If the computed upper bound of $w_v(G)$ is not better than the weight of the best clique found so far, the effort spent in MaxSAT reasoning is useless. However, *GetBranches* does not encode G to MaxSAT and begins IS splitting from a part of G, which is particularly effective on large graphs, because B is generally significantly reduced, even if $w_v(G)$ is greater than the weight of the best clique found so far.

Algorithm 5: WLMC(G), a BnB algorithm for MWC

Input: $G = (V, E, w)$
Output: a maximum weight clique C_{max} of G

```
1  begin
2      (C_0, O_0, G') ← Initialize(G, 0);
3      C_max ← C_0, V' ← the vertex set of G';
4      order V' w.r.t. the initial ordering O_0;
5      for i:= |V'| to 1 do
6          P ← Γ(v_i)∩{v_{i+1},v_{i+2},...,v_{|V'|}};
7          if w(P) + w(v_i) > w(C_max) then
8              (C'_0, O'_0, G'') ←
9                  Initialize(G[P], w(C_max) − w(v_i));
10             if w(C'_0) + w(v_i) > w(C_max) then
11                 C_max ← C'_0∪{v_i};
12             C' ← SearchMaxWClique(G'', C_max,
13                                   {v_i}, O'_0);
14             if w(C') > w(C_max) then C_max ← C';
15     return C_max;
```

Algorithm WLMC

Algorithm 5 describes WLMC. It combines the procedures *Initialize* and *SearchMaxWClique*. Roughly speaking, WLMC calls *Initialize* to preprocess both the input G and the first-level subgraphs in the search tree, and then calls *SearchMaxWClique* to recursively search for a maximum weight clique in the reduced subgraphs.

WLMC first calls $Initialize(G, 0)$ (the initial lb of $\omega_v(G)$ is 0) to derive an initial clique C_0, an initial ordering O_0 and a reduced subgraph G', and instantiates C_{max} with the initial clique C_0. Then, WLMC unrolls the first level subgraphs induced by the set of candidates $\Gamma(v_i)\cap\{v_{i+1}, \ldots, v_{|V'|}\}$, denoted by P, for $i = |V'|$ to 1 respecting the initial vertex ordering O_0. If $w(P)$+$w(v_i)$ is not greater than $w(C_{max})$, then a clique of weight greater than $w(C_{max})$ cannot be found in $G[P]$ and the search in $G[P]$ is pruned. Otherwise, WLMC calls $Initialize(G[P], w(C_{max})\text{-}w(v_i))$ to compute an initial clique C'_0 of $G[P]$, a vertex ordering O'_0 and a reduced subgraph G'' of $G[P]$. Finally, *SearchWMaxClique* is called to recursively search for a clique C' containing v_i, of weight greater than $w(C_{max})$, in the subgraph G'', and updates C_{max} with C' if $w(C')$ is greater than $w(C_{max})$.

Since G is large, the first level subgraphs may still contain a lot of vertices. With a growing lower bound $w(C_{max})$ of $\omega_v(G)$, the first level subgraphs can be further reduced, which is useful to speed up the search in *SearchWMaxClique*. Moreover, re-ordering the vertices in the subgraphs near the root of the search tree was showed to be very effective in BnB MC algorithms (Konc and Janezic 2007). This is the rationale behind also applying *Initialize* to the first-level subgraphs.

Empirical Investigation

We empirically evaluated WLMC and compared it with some of the most competitive exact and heuristic MWC algorithms (also called *solvers*). WLMC 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. Experiments were performed on an AMD Opteron CPU 2435@2.6GHz under Linux with 32GB memory.

In the experiments, we compared the following solvers:

Cliquer: It is an exact solver for both MC and MWC (Ostergard 2001; 2002). We used its latest version (http://users.tkk.fi/pat/cliquer.html), released in 2010.

MWCLQ: It is one of the best exact MWC solvers, which applies a MaxSAT reasoning variant to compute a tighter UB of $\omega_v(G)$ at each search tree node (Fang, Li, and Xu 2016) to reduce the search space.

LSCC+BMS: It is a very recent heuristic MWC solver, which uses a heuristic, called Best from Multiple Selection (BMS), to improve the performance in large sparse graphs (Wang, Cai, and Yin 2016).

FastWClq: It is a yet more recent heuristic MWC solver, which interleaves between clique construction and graph reduction, and can prove the optimality of its solutions in some cases (Cai and Lin 2016).

The source code of the last three solvers was provided by their authors, and compiled using their Makefiles.

In the first experiments, we considered 187 real-world graphs from the Network Data Repository (Rossi and Nesreen 2015), available at http://networkrepository.com, including the 86 and 90 graphs used to evaluate LSCC+BMS and FastWClq in (Wang, Cai, and Yin 2016; Cai and Lin 2016). Weights were assigned to vertices as in (Cai and Lin 2016). For WLMC, LSCC+BMS and FastWClq, Table 1 shows their best solutions and runtimes in seconds (including the preprocessing and search times, not including the time for reading the input graphs). For heuristic solvers, 10 independent runs with different seeds were performed for each graph, each run finding a solution sol that is the best in this run. The mean time (avgt.) to reach sol over the 10 runs, as well as the best quality sol (best) over the 10 runs, is showed. The cutoff time was set to 1000s, except for 6 hard graphs whose limit was 5 hours.

For lack of space, we exclude 135 graphs that are solved by WLMC within 3s and report results for the remaining 52 graphs, whose number of vertices ranges from 8K to 59M. The best times are in bold (for heuristic solvers, times are not in bold if the best weight found is not the optimum).

WLMC finds and proves the optimum for all the graphs. Among the 52 instances reported in Table 1, FastWClq proves the optimum for 8 instances. Nevertheless, LSCC+BMS and FastWClq do not find the optimum on 31 and 21 instances, respectively. For *friendster*, whose optimum is 5511, the best solution is 2885 for both heuristic solvers; and for *soc-sinaweibo*, whose optimum is 4759, the best solutions of LSCC+BMS and FastWClq are 3555 and 1424, respectively. For the 6 hardest instances, LSCC+BMS and FastWClq do not find any optimum in 5h.

In terms of runtimes, WLMC needs less time on 44 instances. For *soc-dogster*, WLMC needs 8.03s, which is 41 and 72 times faster than LSCC+BMS (332.9s) and FastWClq (585.3s), and for *dbpedia-link*, WLMC needs 54.97s, which is 8 and 15 times faster than LSCC+BMS (442.7s) and FastWClq (839.3s). Moreover, the heuristic solvers fail to find the optimum of these graphs. In general, WLMC is faster than LSCC+BMS and FastWClq. These results indicate that WLMC is an extremely competitive exact solver.

We also compared WLMC with two exact solvers: Cliquer and MWCLQ. While WLMC solved all the 187 graphs, MWCLQ did not find any optimum for the graphs in Table 1 and Cliquer only found 5 optimums (*rec-dating*, *rec-libimseti-dir*, *rec-movielens*, *scc_twitter-copen*, *sc-TSOPF-RS-b2383-c1*) for the graphs in Table 1. Although Cliquer and MWCLQ are efficient on small and medium graphs, they are not suitable for large graphs.

We evaluated the impact of preprocessing the first level subgraphs and of incremental vertex-weight splitting in WLMC by comparing it with the following variants:

WLMC\prep1: It is WLMC without preprocessing the first level subgraphs. Line 8 in Algorithm 5, which calls the procedure *Initialize*, is removed.

WLMC\UP&Split: It is WLMC without incremental vertex-weight splitting. UP&Split is removed from GetBranches (Algorithm 3).

Table 2 shows the search tree size and the search time of WLMC, WLMC\prep1 and WLMC\UP&Split on the graphs of Table 1 with solving times beyond 40s, and using the same cutoff times. With preprocessing and incremental vertex-weight splitting, the search tree size of WLMC is almost always the smallest. The search time of WLMC is comparable with that of WLMC\prep1 and WLMC\UP&Split on easy graphs. However, WLMC is substantially faster than WLMC\prep1 and WLMC\UP&Split on hard graphs. In particular, both WLMC\prep1 and WLMC\UP&Split fail to solve the 6 hard graphs within the cutoff times. In addition, WLMC is 13 and 18 times faster than WLMC\prep1 and WLMC\UP&Split, respectively, for *soc-flickr-und*.

Table 3 shows the effect of the procedure $Initialize$. For each graph G of Table 2, Table 3 reports $\omega_v(G)$, the weight of the initial clique C_0 found by $Initialize$ at the root of the search tree (line 2 of Algorithm 5), the ratio rt of the number of vertices in the reduced graph G' to the number of vertices in the original graph, and the mean ratio rt' of the number of vertices in the reduced graphs G'' of the first level (line 8

Table 1: Comparison of WLMC with two heuristic algorithms LSCC+BMS and FastWClq. The optimum and best times are in bold. '-' means that LSCC+BMS or FastWClq did not find the displayed solution in all runs.

Instance	WLMC		LSCC+BMS		FastWClq	
#cutoff=1000s	$\omega_v(G)$	time	best	avgt.	best	avgt.
aff-flickr-user-groups	**1720**	**9.23**	**1720**	10.95	1640	371.1
aff-orkut-user2groups	**971**	**764.7**	965$^-$	373.0	831	445.4
bn-human-BNU_1_00 25865_session_2-bg	**19189**	204.6	13604$^-$	571.7	**19189**	**105.7**
channel-500x100 x100-b050	**796**	13.47	**796**	3.75	**796**	**1.04**
dbpedia-link	**5062**	**54.97**	4396$^-$	442.7	4156$^-$	839.3
delaunay_n22	**796**	**3.36**	793$^-$	172.9	**796**	3.42
delaunay_n23	**798**	**5.24**	794$^-$	481.9	**798**	6.71
delaunay_n24	**797**	**11.86**	790$^-$	461.5	**797**	12.95
friendster	**5511**	**13.02**	2885$^-$	437.8	2885	134.6
hugebubbles-00020	**400**	**9.87**	**400**$^-$	83.62	**400**	10.37
hugetrace-00010	**400**	**4.75**	399	18.42	**400**	5.42
hugetrace-00020	**400**	7.72	**400**$^-$	160.0	**400**	**7.19**
inf-europe_osm	**646**	**13.24**	594$^-$	280.4	**646**	15.12
inf-germany_osm	**597**	4.44	579$^-$	550.0	**597**	**3.65**
inf-road-usa	**766**	**11.51**	597$^-$	423.9	**766**	14.73
rec-dating	**1699**	37.42	**1699**	**1.50**	1459	612.7
rec-epinions	**1054**	**10.37**	**1054**	20.59	1028	594.4
rec-libimseti-dir	**1938**	32.17	**1938**	**3.26**	1768	415.5
rec-movielens	**3777**	63.57	**3777**	**10.64**	3288$^-$	740.4
rgg_n_2_23_s0	**2407**	**10.61**	2192$^-$	574.2	**2407**	16.60
rgg_n_2_24_s0	**2514**	**23.75**	2177$^-$	373.5	**2514**	64.84
scc_twitter-copen	**58699**	95.89	**58699**	2.50	**58699**	**0.43**
sc-rel9	**572**	**3.43**	**572**$^-$	122.4	**572**	21.35
sc-TSOPF-RS -b2383-c1	**960**	**55.46**	**960**$^-$	513.9	**960**$^-$	897.2
soc-BlogCatalog	**4803**	**6.13**	**4803**$^-$	325.5	**4803**	170.1
soc-buzznet	**2981**	**3.90**	**2981**	31.21	**2981**	98.67
soc-digg	**5303**	**10.23**	5283$^-$	549.6	**5303**	113.8
soc-dogster	**4418**	**8.03**	4356$^-$	332.9	4404$^-$	585.3
socfb-A-anon	**2872**	**9.99**	**2872**$^-$	400.2	**2872**	57.75
socfb-B-anon	**2662**	**9.30**	2620$^-$	324.7	**2662**	112.9
socfb-uci-uni	**1045**	**27.30**	995$^-$	642.1	**1045**	110.0
soc-flickr	**7083**	**8.94**	7050$^-$	298.7	**7083**	38.94
soc-flickr-und	**10127**	**329.8**	9935$^-$	311.0	10115	921.3
soc-livejournal	**21368**	**3.25**	17375$^-$	532.7	**21368**	14.28
soc-livejournal-user-groups	**1054**	**133.1**	**1054**$^-$	440.6	878$^-$	849.5
soc-ljournal-2008	**40432**	**14.96**	37363$^-$	368.6	**40432**	75.40
soc-orkut-dir	**6147**	**95.34**	6084$^-$	629.5	**6147**	133.5
soc-orkut	**5452**	**100.2**	**5452**$^-$	524.6	**5452**	120.4
soc-pokec	**3191**	**7.19**	**3191**$^-$	592.8	**3191**	8.02
soc-sinaweibo	**4759**	**90.57**	3555$^-$	469.2	1424$^-$	703.5
soc-twitter-higgs	**8039**	**6.81**	**8039**$^-$	305.2	5383	276.2
tech-ip	**668**	**17.47**	**668**$^-$	573.8	123	1.03
web-baidu-baike	**3814**	**7.73**	2651$^-$	486.8	**3814**	101.9
web-wikipedia-growth	**4741**	**22.69**	**4741**$^-$	449.9	**4741**	269.0
web-wikipedia_link_it	**89947**	**190.3**	**89947**$^-$	322.3	2202	115.3
wikipedia_link_en	**4624**	**14.77**	1856$^-$	488.8	**4624**	195.5
#cutoff=5h for the following 6 hard instances						
aff-digg	**3836**	**1288**	3776$^-$	5371	3353$^-$	11882
bio-human-gene1	**134713**	**15686**	134292$^-$	9314	134362$^-$	7273
bio-human-gene2	**135310**	**13226**	135152$^-$	9018	135059$^-$	1768
bio-mouse-gene	**59952**	**13787**	59921$^-$	7112	59855$^-$	2930
bn-human-BNU_1_0 025865_session_1-bg	**20598**	**2237**	19539	2845	20214$^-$	8441
twitter_mpi	**13524**	**4117**	12939$^-$	7162	12145$^-$	16218

Table 2: Search tree sizes in thousands and search times in seconds of WLMC, WLMC\prep1 and WLMC\UP&Split.

Instance	WLMC		WLMC \prep1		WLMC \UP&Split	
	tree	time	tree	time	tree	time
aff-digg	**20459**	**1288**	26868	1670	156682	1520
aff-orkut-user2groups	3458	764.7	**3446**	**596.8**	3470	859.6
bio-human-gene1	**2601**	**15686**	-	-	-	-
bio-human-gene2	**2817**	**13226**	-	-	-	-
bio-mouse-gene	**5722**	**13787**	-	-	-	-
bn-human-BNU_1_0025865_session_1-bg	**786.1**	**2237**	-	-	-	-
bn-human-BNU_1_0025865_session_2-bg	**421.0**	**204.6**	-	-	-	-
dbpedia-link	**617.1**	54.97	622.7	**43.36**	656.5	60.54
rec-movielens	**427.2**	63.57	635.2	58.12	1752	**54.27**
sc-TSOPF-RS-b2383-c1	237.7	55.46	**163.1**	56.66	245.3	**32.77**
scc_twitter-copen	**217.2**	95.89	240.9	107.8	**217.2**	**42.41**
soc-flickr-und	**741.1**	**329.8**	5570	4274	212938	5970
soc-livejournal-user-groups	**2344**	133.1	2355	**99.20**	2594	110.5
soc-orkut	**613.2**	100.2	631.6	**66.83**	741.0	89.38
soc-orkut-dir	**538.9**	95.34	608.5	**77.63**	773.2	88.15
soc-sinaweibo	**667.4**	**90.57**	711.2	94.08	710.2	99.22
twitter_mpi	**4133**	**4117**	-	-	-	-
web-wikipedia_link_it	**203.4**	190.3	**203.4**	195.8	**203.4**	**98.58**

Table 3: The effect of preprocessing: C_0 is the initial clique found at the root, rt is the ratio of the number of vertices in the reduced G' at the root to the number of vertices in the original G, and rt' is the mean ratio of the number of vertices in the reduced G'' at the first level to the number of vertices in $G[P]$.

Instance	$\omega_v(G)$	$w(C_0)$	rt	rt'
aff-digg	3836	2721	0.168	0.380
aff-orkut-user2groups	971	213	0.849	0.019
bio-human-gene1	134713	131692	0.283	0.949
bio-human-gene2	135310	132080	0.353	0.953
bio-mouse-gene	59952	44472	0.403	0.573
bn-human-BNU_1_00 25865_session_1-bg	20598	16749	0.074	0.517
bn-human-BNU_1_00 25865_session_2-bg	19189	8411	0.054	0.463
dbpedia-link	5062	1152	0.186	0.022
rec-movielens	3777	2167	0.949	0.381
sc-TSOPF-RS-b2383-c1	960	415	0.994	0.010
scc_twitter-copen	58699	57995	0.090	0.918
soc-flickr-und	10127	6698	0.040	0.292
soc-livejournal-user-groups	1054	489	0.413	0.038
soc-orkut	5452	1898	0.756	0.071
soc-orkut-dir	6147	1170	0.862	0.079
soc-sinaweibo	4759	834	0.126	0.026
twitter_mpi	13524	9101	0.033	0.192
web-wikipedia_link_it	89947	89539	0.003	0.987

of Algorithm 5) to the number of vertices in $G[P]$. We can see that the quality of C_0 is good for several graphs and that these graphs are significantly reduced at the root and the first level of the search tree by the procedure $Initialize$.

We conducted additional experiments and compared the quality of the solution found by WLMC, MWCLQ, Cliquer, LSCC+BMS and FastWClq for large DIMACS graphs[1] within 3600 seconds. These graphs have more than 1000 vertices. The original graphs are not weighted. We assign weights to their vertices as in (Cai and Lin 2016). Table 4 shows the results. LSCC+BMS generally finds better solutions than the other algorithms. WLMC is comparable to MWCLQ and FastWClq, because WLMC finds better solution than FastWClq and MWCLQ for five and six graphs respectively, while FastWclq and MWCLQ find better solutions than WLMC for six and four graphs respectively. Nevertheless, while LSCC+BMS and FastWClq find an optimal solution for six graphs, WLMC finds and proves an optimal solution for seven graphs.

Table 4: Solution quality for DIMACS graphs with more than 1000 vertices within 3600 seconds. Optimums marked with '*'.

Instance	WLMC	MWCLQ	Cliquer	LSCC+BMS	FastWClq
C1000.9	7341	8471	1165	**9072**	8552
C2000.5	**2466**	**2466**	**2466**	**2466**	2449
C2000.9	7862	10034	529	**10333**	9516
C4000.5	2438	2698	1233	**2792**	2542
DSJC1000_5	**2186***	**2186***	**2186***	**2186***	**2186***
MANN_a45	**34265***	34133	1276	34183	34121
MANN_a81	**111139**	111033	195	111094	110481
hamming10-2	**50512***	**50512***	**50512***	**50512***	**50512***
hamming10-4	4812	4614	735	**5129**	4990
keller6	4760	6316	511	**7360**	5772
p_hat1000-1	**1514***	**1514***	**1514***	**1514***	**1514***
p_hat1000-2	**5777***	**5777***	5612	**5777***	**5777***
p_hat1000-3	8086	7588	2417	**8111**	7967
p_hat1500-1	**1619***	**1619***	**1619***	**1619***	**1619***
p_hat1500-2	**7360**	7104	2897	**7360**	7355
p_hat1500-3	9846	8449	1497	**10278**	9875
san1000	**1716***	**1716***	929	**1716***	**1716***

An important application of MWC is to solve the winner determination problem (WDP) in combinatorial auctions, because WDP can naturally be formulated as MWC (Fang, Li, and Xu 2016). We compared WLMC with MWCLQ, Cliquer, LSCC+BMS and FastWClq on the WDP benchmark provided in (Lau and Goh 2002), which has been widely used to test WDP algorithms (Fang, Li, and Xu 2016). The benchmark contains 500 instances with up to 1500 items and 1500 bids, and can be divided into 5 groups by the item number and the bid number. Each group contains 100 instances labeled as REL–m–n, where m is the number of items and n is the number of bids. When formulated as MWC, the graphs contain up to 1500 vertices with density from 0.06 to 0.33.

Table 5 shows the average performance of the five MWC solvers for the five groups of the WDP instances. WLMC and MWCLQ are the only solvers able to quickly find and prove the optimal solution of all the instances. The two heuristic algorithms LSCC+BMS and FastWClq cannot find any optimal solution for some graphs within the cutoff time.

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

Table 5: Mean runtimes in seconds for five groups of WDP instances, '#' stands for the number of instances for which an optimal solution is found by a solver within 500 seconds.

Group	WLMC		MWCLQ		Cliquer		LSCC +BMS		Fast WClq	
	#	time	#	time	#	time	#	time	#	time
REL-500-1000	100	117	100	**97.1**	20	377	88	101	0	-
REL-1000-1000	100	2.81	100	**2.44**	100	5.58	100	4.62	77	103
REL-1000-500	100	**0.11**	100	0.12	100	**0.11**	100	0.23	100	2.08
REL-1000-1500	100	**2.26**	100	2.48	100	4.46	100	8.25	78	130
REL-1500-1500	100	**3.24**	100	3.71	100	5.14	100	5.00	83	108

Conclusions

We proposed WLMC, a new exact MWC algorithm that is very effective on large graphs because it combines an efficient preprocessing and incremental vertex-weight splitting in a BnB scheme. WLMC greatly outperforms relevant heuristic and exact solvers on practical instances, and the reported results refute the prevailing hypothesis that exact algorithms are less adequate for large graphs.

Acknowledgements

This work is supported by NSFC Grants No. 61272014, No. 61370183, No. 61472147 and No. 61370184, the MeCS platform of the University of Picardie Jules Verne and the HPC platform of Jianghan Univeristy. The third author was supported by Mobility Grant PRX16/00215 of the Ministerio de Educación, Cultura y Deporte, the Generalitat de Catalunya grant AGAUR 2014-SGR-118, and the MINECO-FEDER project RASO TIN2015-71799-C2-1-P.

References

Cai, S. W., and Lin, J. K. 2016. Fast solving maximum weight clique problem in massive graphs. In *Proceedings of 25th International Joint Conference on Artificial Intelligence, IJCAI*, 568–574.

Carraghan, R., and Pardalos, P. 1990. An exact algorithm for the maximum clique problem. *Operations Research Letters* 9(6):375–382.

Fang, Z. W.; 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* 55:1–35.

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.

Konc, J., and Janezic, D. 2007. An improved branch and bound algorithm for the maximum clique problem. *Communications in Mathematical and in Computer Chemistry* 58(3):569–590.

Kumlander, D. 2004. A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. In *Proceedings of Modelling, Computation and Optimization in Information Systems and Management Sciences, MCO*, 202–208. Hermes Science Publishing.

Kumlander, D. 2008. On importance of a special sorting in the maximum-weight clique algorithm based on colour classes. In *Proceedings of the 2nd International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences, MCO*, volume 14 of *Communications in Computer and Information Science*, 165–174. Springer.

Lau, H. C., and Goh, Y. G. 2002. An intelligent brokering system to support multi-agent web-based 4 th-party logistics. In *Proceedings of the14th IEEE International Conference on Tools with Artificial Intelligence, ICTAI*, 154–161. IEEE Press.

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.; Zhu, Z.; Manyà, F.; and Laurent, S. 2012. Optimizing with minimum satisfiability. *Artificial Intelligence* 190:32–44.

Ma, T. Y., and Latecki, L. J. 2012. Maximum weight cliques with mutex constraints for video object segmentation. In *Proceedings of 2012 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2012*, 670–677.

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 Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, 1274–1279.

Newman, M. 2003. The structure and function of complex networks. In *SIAM Rev*, 167–256.

Ostergard, P. 2001. A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing* 8(4):424–436.

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

Rossi, R., and Nesreen, K. 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.

Rossi, R.; Gleich, D.; Gebremedhin, A.; and Patwary, M. 2013. Parallel maximum clique algorithms with applications to network analysis and storage. *Eprint Arxiv*.

San Segundo, P.; Alvaro, L.; and Pardalos, P. 2016. A new exact maximum clique algorithm for large and massive sparse graphs. *Computers & Operations Research* 66:81–94.

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.

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

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

Zhang, D.; Javed, O.; and Shah, M. 2014. Video object cosegmentation by regulated maximum weight cliques. In *Proceedings of European Conference on Computer Vision, ECCV*, 551–566. Springer.

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.