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 $\omega_w(G)$.

MWC is NP-Hard and has practical applications in different domains such as protein structure prediction (Mascia et al. 2010), coding theory (Zhan 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 Fast-WClq (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), OT-Clique (Shimizu et al. 2013), and MWC/LQ (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
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 **SearchMaxWCl**ique 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 $v_i$ 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 shown 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 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 **SearchMaxWCl**ique**

Given a graph $G = (V, E, w)$, the best clique $C_{max}$ so far and an ordering $O$ over $V$, the BnB procedure **SearchMaxWCl**ique 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 $U \leftarrow V$
3 compute the degree $deg(v)$ for each vertex $v$ in $U$
4 for $i := 1$ to $|V|$ do
5 $v_i \leftarrow$ the vertex $v$ with the smallest $deg(v)$ in $U$
6 if $deg(v_i) = |U| - 1$ then
7 $I = \{v_i\}$ is adjacent to all other vertices in $U$ */
8 order $U$ arbitrarily as $\{v_1, v_2, \ldots, v_{|V|}\}$;
9 $C_0 \leftarrow U$; break;
10 $U \leftarrow U \setminus \{v_i\}$
11 for each neighbor $v_i$ of $v_i$ in $U$ do
12 $deg(v) \leftarrow deg(v) - 1$
13 if $w(C_0) > lb$ then $lb \leftarrow w(C_0)$
14 let $w^*(v)$ denote the total weight of $v$ and its neighbors in $G$
15 $G' \leftarrow G$ after removing vertices $v$ s.t. $w^*(v) \leq lb$
16 $O_0 \leftarrow v_1 < v_2 < \cdots < v_{|V|}$;
17 return $(C_0, O_0, G')$
18 end

Algorithm 2 shows the pseudo-code of **SearchMaxWCl**ique. 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 $G[\Gamma(b)] \cap (\{b_1, b_2, \ldots, b_{|B|}\} \cup A)$, 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 $\Pi$. 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 $w_i$ denotes vertex $v_i$ with weight $w_i = w(v_i)$,
Algorithm 2: SearchMaxWClique(G, Cmax, C, O)

Input: G = (V, E, w), the best clique Cmax so far, the current growing clique C, a vertex ordering O
Output: Clique C if w(C) > w(Cmax); otherwise Cmax
1 begin
2 if |V| = 0 then return C;
3 B ← GetBranches(G, w(Cmax) − w(C), O);
4 if B = ∅ then return Cmax;
5 A ← V \ B;
6 Let B = {b1, b2, ..., b|B|}, b1 < b2 < ... < b|B| w.r.t. O;
7 for i := |B| to 1 do
8 P ← Γ(bi) ∩ {b1+1, b1+2, ..., b|B|} \ A;
9 if w(C ∪ {bi}) + w(P) > w(Cmax) then
10 C’ ← SearchMaxWClique(G[P], Cmax, C ∪{bi}, O);
11 if w(C’) > w(Cmax) then Cmax ← C’;
12 return Cmax;
13 end

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 ← ∅; I ← ∅; /* II 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 ∈ II s.t. Γ(v) ∩ D = ∅ and
7 \[\sum_{j=1}^{||II||} w_{max}(D_j) \leq t \] after adding v into D
8 then D ← D ∪ {v};
9 else if \[\sum_{j=1}^{||II||} w_{max}(D_j) + w(v) \leq t \]
10 create a new IS D = {v}, II ← II ∪ {D};
11 else B ← B ∪ {v};
12 ub0 ← \[\sum_{j=1}^{||II||} w_{max}(D_j)\];
13 Let B = {b1, b2, ..., b|B|}, b1 < b2 < ... < b|B| w.r.t. O;
14 for i := |B| to 1 do
15 (ub, II) ← UP&PSplit(G, II ∪ {b1});
16 ub0 + w(bi), i);
17 if ub ≤ t then
18 ub0 ← ub, II ← II’, B ← B \ {bi};
19 return B;
20 end

and let O: v1 < v2 < ... < v6 be the vertex ordering. Assume the best clique weight so far is w(Cmax) = 6,
and we call GetBranches(G, 6, O). During the first phase, GetBranches inserts the vertices v1, v2, v3, v4, v5, v6 into two ISs: 
D1 = {v1, v2, v3} and D2 = {v4, v5, v6}. Afterwards, v3 has adjacent vertices in both D1 and D2, and GetBranches cannot
create a new IS D3 = {v3} because then \[\sum_{j=1}^{3} w_{max}(D_j) > 6\]. Hence, the first phase gives

A = \{v6, v4, v5, v2, v3\} and B = \{v1\}.

The second phase of GetBranches (lines 12–18) tries to remove each vertex bi ∈ B from B and insert it into A. Recall that II = \{D1, D2, ..., D|II|\} and A = V(II). In order to insert bi into A, we have to show that G[A ∪ {bi}] does not contain any clique of weight greater than t. Since any clique in G[A ∪ {bi}] is formed by at most one vertex from each IS of II and possibly by bi, an upper bound of
its weight is \[\sum_{j=1}^{t|II|} w_{max}(D_j) + w(b_i)\]. 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 II
and by bi. 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, bi 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 II = \{D1, D2\}, D1 = \{v6, v4, v5\}, D2 = \{v2, v3\}, B = \{v1\} and t = 6 (cf. Example 1). Our objective here is
to show that the vertices in II ∪ \{v3\} cannot form any clique of weight greater than 6.

The initial upper bound for II ∪ \{v3\} is \[w_{max}(D_1) + w_{max}(D_2) + w(v_3)\] = 9. If v3 is in a clique C of G, then v1, v2, and v3 cannot be in C because they are not adjacent
to v1. So, we remove v3 from D2, and v2 and v3 from D1. Since D1 becomes unit, its unique vertex v2 is added to C, which
removes v3 from D2 because v3 is not adjacent to v2, and D2 becomes empty. This reasoning shows that if v3 is in
a clique C and D1 contains a vertex in C, then D2 cannot contain any vertex in C. So, D1, D2, and D3 are conflicting,
because one of these ISs cannot contain any vertex in C. 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 contains the most weighted vertices v2 and v1, which is
impossible because v2 and v1 are not adjacent in G. A further improvement of the upper bound can be obtained by
splitting all the weights greater than 2 in D1, D2, and \{v3\}. The splitting gives a set of ISs II1 = \{D1, D2, \{v3\}\}. The splitting
in each IS is 2, and a set of

Figure 1: A graph with \(\omega_6(G)=6\)
ISs $\Pi_R = \{\{v_2^2, v_3^1\}, \{v_1^1\}\}$ consisting of the weights split from $\Pi$. Note that the splitting of $\Pi$ 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 $C_G$, ($C_{G_R}$). 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 $\Pi_1$ partitions $G_1$ and is conflicting, for any clique in $G_1$, there is an IS of $\Pi_1$ such that the clique does not contain any vertex from the IS. So, $UB_{G_1} = 2 + 2 + 2 = 4$.

$\Pi_R = \{\{v_1^1\}, \{v_2^1, v_1^2\}\}$ partitions $G_R$ after removing all the vertices with weight 0. If $v_1^1$ is a clique, then the most weighted vertex $v_2^1$ in $\{v_2^1, v_1^2\}$ cannot be in the clique, because $v_1^1$ and $v_2^1$ are not adjacent. Consequently, $\{v_2^1, v_1^2\}$ can be split into $\{v_2^1\}$ and $\{v_1^2\}$, so that $\{v_1^1\}$ and $\{v_2^1\}$ 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 $\Pi_2 = \{\{v_1^2\}, \{v_2^1\}\}$ and $G_3$ can be partitioned into $\Pi_3 = \{\{v_2^1\}, \{v_1^2\}\}$, 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, $UB_{G_1} = UB_{G_3} = 1$. So $w(C_G) \leq 4 + 1 + 1 = 6$, meaning that $v_1^1$ 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 $\{S_0, S_1, S_2, \ldots, S_l\}$. Let $\delta = \min(w_{\max}(S_0), \ldots, w_{\max}(S_l))$, we split each weight greater than $\delta$ 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 $\{v_0^1, v_0^2, v_1^1\}$ is split into $S_j' = \{v_0^1, v_0^2, v_1^2\}$ and $S_j'' = \{v_0^2, v_1^2\}$ by splitting all the weights $\delta$ greater than $\delta$ and $\delta$ and by keeping all the weights smaller than or equal to $\delta$ in $S_i'$. Consequently, we obtain a conflicting subset of ISs $\{S_0', S_1', S_2', \ldots, S_l'\}$, in which $w_{\max}(S_j') = \delta$ for each $j$, $0 \leq j \leq r$, and a subset of ISs $\{S_0'', S_1'', S_2'', \ldots, S_l''\}$ from which further conflicts can be detected. The set of conflicting ISs $\{S_0', S_1', S_2', \ldots, S_l'\}$ allows to improve the upper bound by $\delta$, 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 $S_0 = \{v_1^1, v_1^2, v_1^3, v_1^4\}$. Assume that unit IS propagation involving the ISs $S_1$, $S_2$, and $S_3$ removes $v_1^3$ from $S_0$, and we have $h_{\max}(S_1), h_{\max}(S_2), h_{\max}(S_3)) = 2$. We split $S_0$ into $S_0'' = \{v_1^1, v_1^2\}$ and $S'' = \{v_1^3\}$, 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 $S_0''$ should be determined. 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 $\min(4 - 3, w_{\max}(S_1), w_{\max}(S_2), w_{\max}(S_3)) = 3$.

Generally speaking, let $S_0 = \{u_1^1, \ldots, u_k^1, \ldots, u_k^{l_{S_0}}\}$ be an IS in which the $k$ most weighted vertices $u_1^1, \ldots, u_k^{l_{S_0}}$ are removed by unit IS propagation involving the ISs $S_1, S_2, \ldots, S_r$. Without loss of generality, assume that $w_1 \geq \cdots \geq w_k \geq w_{k+1} \geq \cdots \geq w_{l_{S_0}}$. Let $\delta = \min(w_1 - w_{k+1}, w_{\max}(S_1), \ldots, w_{\max}(S_r))$ and let $u_{k+1} = w_j - \min(\delta, w_j - w_{k+1})$ for $1 \leq j \leq k$. We split $S_j$ into $S_j' = \{u_1^1, \ldots, u_{k+1}^1, \ldots, u_k^{l_{S_0}}\}$ and $S_j'' = \{u_1^1, \ldots, u_k^{l_{S_0}}, u_{k+1}^{l_{S_j}}, \ldots, u_{l_{S_0}}^{l_{S_j}}\}$, 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) $\min(\delta, w_1 - w_{k+1}) \geq \min(\delta, w_2 - w_{k+1}) \geq \cdots \geq \min(\delta, w_r - w_{k+1})$ in $S_0''$, (2) $w_1 \geq w_2 \geq \cdots \geq w_k \geq w_{k+1} \geq \cdots \geq w_{l_{S_0}}$ 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 \( 1 \leq j < k \), \( w_j'''' = w_j'' - w_{j+1}'' = w_j - \min(\delta, w_j - w_{j+1}) = w_j - \min(\delta, w_j - w_{j+1}) \geq 0 \).

From (1) and (2), we easily see that \( \max(S_0) + w_{\text{max}}{S_0}' \leq \max(\delta, w_0 - w_1) + w_1'' = \max(\delta, w_0 - w_1) + w_1'' \leq \max(\delta, w_0 - w_1) \).

Theorem 1 is proved.

Algorithm 4 implements the function \( \text{UP\&Split} \) 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 \( \text{split}(S, \delta) \) to split each involved IS \( S \) into two ISs \( S' \) and \( S'' \). Let \( S = \{ u_1^0, \ldots, u_n^0 \} \) be an IS, with \( w_j \geq \cdots \geq w_k \geq \cdots \geq w_0 \) for all \( 0 \leq \delta \leq w_j \). Then, \( \text{split}(S, \delta) \) returns \( S' = \{ u_1^\delta, \ldots, u_k^\delta, u_{k+1}^\delta, \ldots, u_n^\delta \} \) and \( S'' = \{ u_k^{\delta - \delta}, \ldots, u_n^{\delta} \} \). In other words, each weight \( w_i \) greater than \( \delta \) is split into \( \delta \) and \( w_i - \delta \), and each weight \( w_i \) not greater than \( \delta \) remains in \( S \). Observe that \( \max(S) = \max(S') + \max(S'') \).

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

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

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 \( w_\Pi \). Since \( w(C) = \sum_{j=1}^{n_j} w_\Pi(C) \), (Condition 2) and \( w(C) \leq UB_j \), it holds that \( w(C) \leq \sum_{j=1}^{n_j} UB_j \). By Condition 4, \( UB_j \leq (\| \Pi_j \| - 1) \times w_{\text{max}}(S_{j1}) \) for each \( j < p \). Hence, \( w(C) \leq \sum_{j=1}^{n_j}([\| \Pi_j \| - 1] \times w_{\text{max}}(S_{j1})) + \sum_{k=1}^{n_{jk}} w_{\text{max}}(S_{jk}) \). By Condition 3, which is the improved upper bound \( ub \) of \( w(C) \) returned by \( \text{UP\&Split} \) together with \( \Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_p \).

**Algorithm 4: UP\&Split(G, \Pi, 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 \( \Pi \)

1. \( \Delta \leftarrow \emptyset; \)
2. **while** there is a non-marked unit IS \( \{v\} \) in \( \Pi \) **do**
3. remove vertices non-adjacent to \( v \) from their IS;
4. **if** there is a non-marked empty IS \( S_0 \) **then**
5. restore all the removed vertices into their IS;
6. Let \( S_1, S_2, \ldots, S_r \) be the ISs responsible of removing all the vertices of \( S_0 \);
7. \( \delta \leftarrow \min(w_{\text{max}}(S_0), \ldots, w_{\text{max}}(S_r)); \)
8. **for** each IS \( S_j \) in \( \{ S_0, S_1, S_2, \ldots, S_r \} \) **do**
9. \( (S_j', S_j'') \leftarrow \text{split}(S_j, \delta); \)
10. \( \Delta \leftarrow \Delta \cup \{ S_0', S_1', \ldots, S_r' \}; \)
11. \( \Pi \leftarrow \Pi \setminus \{ S_0, S_1, S_2, \ldots, S_r \} \cup \{ S_0'', S_1'', \ldots, S_r'' \}; \)
12. \( ub \leftarrow ub - \delta; \)
13. **else** there is a non-marked IS \( S_0 \) in which the k most weighted vertices are removed **then**
14. Let \( S_0 = \{ u_1^0, \ldots, u_k^0, \ldots, u_n^{|\Pi_0|} \} \) with \( w_0 \geq \cdots \geq w_{k-1} \geq \cdots \geq w_n \); \( S_0'' = \{ u_1', \ldots, u_k', u_{k+1}', \ldots, u_n' \} \);
15. Let \( u_1', \ldots, u_k' \) be the \( k \) most weighted vertices removed from \( S_0 \);
16. \( \beta \leftarrow w_k - w_{k+1}; \)
17. restore all the removed vertices into their IS;
18. Let \( S_1, S_2, \ldots, S_r \) be the ISs responsible of removing \( u_1', \ldots, u_k' \) from \( S_0 \);
19. \( \delta \leftarrow \min(\beta, w_{\text{max}}(S_1), \ldots, w_{\text{max}}(S_r)); \)
20. \( \gamma \leftarrow w_{k+1}; \)
21. \( S_0' \leftarrow \{ u_1'^{\min(\delta, w_1 - \gamma)}, \ldots, u_k'^{\min(\delta, w_k - \gamma)} \}; \)
22. \( \delta' \leftarrow w_j - \min(\delta, w_j - \gamma) \); \( 1 \leq j \leq k; \)
23. \( S_0'' \leftarrow \{ u_1', \ldots, u_k', u_{k+1}', \ldots, u_n' \}; \)
24. **for** each IS \( S_j \) in \( \{ S_1, S_2, \ldots, S_r \} \) **do**
25. \( (S_j', S_j'') \leftarrow \text{split}(S_j, \delta'); \)
26. \( \Delta \leftarrow \Delta \cup \{ S_0', S_1', \ldots, S_r' \}; \)
27. \( \Pi \leftarrow \Pi \setminus \{ S_0, S_1, S_2, \ldots, S_r \} \cup \{ S_0'', S_1'', \ldots, S_r'' \}; \)
28. \( ub \leftarrow ub - \delta; \)
29. **if** \( ub \leq t \) **then**
30. mark all ISs in \( \Delta; \) break;
31. restore all the removed vertices into their IS;
32. **return** \( (ub, \Delta, \Pi) \);
33. **end**

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 \( \Pi_i \cup \{ \{b_i\} \} \). Note that \( \Pi_1 \cup \Pi_2 \cup \ldots \cup \Pi_{i-1} \) 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 \( B \), \( \Pi \) is not transformed, and \( \Pi \cup \{ \{b_i\} \} \) is used for reasoning on \( b_{i-1} \).

GetBranches performs IS splitting incrementally. It first performs IS splitting in \( \Pi_1 \cup \{ \{b_i\} \} \), obtaining an improved
upper bound \(ub\) and a set \(\Pi'\) of ISs; If \(ub < t\), it performs IS splitting in \(\Pi' \cup \{\{b_{[1,1-1]\}}\}\), 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 \(\phi\) and performs MaxSAT reasoning to split the soft clauses in \(\phi\) using two sophisticated inference rules called \(\delta\)-rule and \((k, \delta)\)-rule, to compute an upper bound of \(w_\Gamma(G)\). 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 \(\phi\) and considers simultaneously all the clauses of \(\phi\). If the computed upper bound of \(w_\Gamma(G)\) 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 \(w_\Gamma(G)\) 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 \(w_{\text{max}}\) of \(G\)

1. \((C_0, O_0, G') \leftarrow \text{Initialize}(G, 0)\);
2. \(C_{\text{max}} \leftarrow C_0, V' \leftarrow \text{the vertex set of } G'\);
3. order \(V'\) w.r.t. the initial ordering \(O_0\);
4. for \(i := |V'|\) to 1 do
5. 5. \(P \leftarrow \Gamma(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_{|V'|}\}\);
6. 6. if \(w(P) + w(v_i) > w(C_{\text{max}})\) then
7. 7. \((C'_0, O'_0, G'') \leftarrow \text{Initialize}(G[P], w(C_{\text{max}}) - w(v_i));
8. 8. if \(w(C'_0) + w(v_i) > w(C_{\text{max}})\) then
9. 9. \(C_{\text{max}} \leftarrow C'_0 \cup \{v_i\}\);
10. 10. \(C' \leftarrow \text{SearchMaxWClq}(G'', C_{\text{max}}, \{v_i\}, O'_0)\);
11. 11. if \(w(C') > w(C_{\text{max}})\) then \(C_{\text{max}} \leftarrow C'\);
12. end

**Return** \(C_{\text{max}}\).

---

### Algorithm WLMC

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

WLMC first calls Initialize(\(G, 0\)) (the initial \(lb\) of \(w_\Gamma(G)\) is 0) to derive an initial clique \(C_0\), an initial ordering \(O_0\) and a reduced subgraph \(G'\), and instantiates \(C_{\text{max}}\) 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 \(w(C_{\text{max}})\), then a clique of weight greater than \(w(C_{\text{max}})\) cannot be found in \(G[P]\) and the search in \(G[P]\) is pruned. Otherwise, WLMC calls Initialize(\(G[P], w(C_{\text{max}})-w(v_i)\)) to compute an initial clique \(C'_0\) of \(G[P]\), a vertex ordering \(O'_0\) and a reduced subgraph \(G''\) of \(G[P]\). Finally, SearchMaxWClq is called to recursively search for a clique \(C'\) containing \(v_i\), of weight greater than \(w(C_{\text{max}})\), in the subgraph \(G''\), and updates \(C_{\text{max}}\) with \(C'\) if \(w(C')\) is greater than \(w(C_{\text{max}})\).

Since \(G\) is large, the first level subgraphs may still contain a lot of vertices. With a growing lower bound \(w(C_{\text{max}})\) of \(w_\Gamma(G)\), the first level subgraphs can be further reduced, which is useful to speed up the search in SearchMaxWClq. 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:

**Clique:** It is an exact solver for both MC and MWC (Osterberg 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 \(w_\Gamma(G)\) 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 interleaves between clique construction and graph reduction, and can prove the optimality of its solutions in some cases (Cai and Lin 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 Nesson 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 (avg.) 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, FastWCq proves the optimum for 8 instances. Nevertheless, LSCC+BMS and FastWCq 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-sinaiveibo, whose optimum is 4759, the best solutions of LSCC+BMS and FastWCq are 3555 and 1424, respectively. For the 6 hardest instances, LSCC+BMS and FastWCq 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 FastWCq (585.3s), and for dpbedia-link, WLMC needs 54.97s, which is 8 and 15 times faster than LSCC+BMS (442.7s) and FastWCq (839.3s). Moreover, the heuristic solvers fail to find the optimum of these graphs. In general, WLMC is faster than LSCC+BMS and FastWCq. 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-TSOPFRS-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>
<thead>
<tr>
<th>Instance</th>
<th>#solves=1000s</th>
<th>WLMC</th>
<th>LSCC+BMS</th>
<th>FastWCq</th>
</tr>
</thead>
<tbody>
<tr>
<td>aff-digg</td>
<td>3836 1288</td>
<td>3736 537</td>
<td>3353 1182</td>
<td></td>
</tr>
<tr>
<td>bio-human-gene1</td>
<td>134713 15686</td>
<td>134292 9314</td>
<td>134362 7273</td>
<td></td>
</tr>
<tr>
<td>bio-human-gene2</td>
<td>135310 13226</td>
<td>135152 9018</td>
<td>135059 1768</td>
<td></td>
</tr>
<tr>
<td>bio-mouse-gene</td>
<td>59952 13787</td>
<td>59921 7112</td>
<td>59855 2930</td>
<td></td>
</tr>
<tr>
<td>bn-human-BNU_1.0</td>
<td>20598 2337</td>
<td>19539 2845</td>
<td>20214 8441</td>
<td></td>
</tr>
<tr>
<td>twitter_npi</td>
<td>13524 4117</td>
<td>12939 7162</td>
<td>12145 16218</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparison of WLMC with two heuristic algorithms LSCC+BMS and FastWCq. The optimum and best times are in bold. '-' means that LSCC+BMS or FastWCq did not find the displayed solution in all runs.
Table 2: Search tree sizes in thousands and search times in seconds of WLMC, WLMC\_\text{\textbackslash prepl} and WLMC\_\text{\textbackslash UP\&Split}.

<table>
<thead>
<tr>
<th>Instance</th>
<th>WLMC \text{\textbackslash prepl}</th>
<th>WLMC \text{\textbackslash UP&amp;Split}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>tree time</td>
<td>tree time</td>
</tr>
<tr>
<td>aff-digg</td>
<td>20459</td>
<td>1288</td>
</tr>
<tr>
<td>aff-orkut-user2groups</td>
<td>3458</td>
<td>764.7</td>
</tr>
<tr>
<td>bio-human-gene1</td>
<td>2601</td>
<td>15686</td>
</tr>
<tr>
<td>bio-human-gene2</td>
<td>2817</td>
<td>13226</td>
</tr>
<tr>
<td>bio-mouse-gene</td>
<td>5722</td>
<td>13787</td>
</tr>
<tr>
<td>bn-human-BNU1</td>
<td>786.1</td>
<td>2237</td>
</tr>
<tr>
<td>0025865_session1_bg</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bn-human-BNU1</td>
<td>421.0</td>
<td>204.6</td>
</tr>
<tr>
<td>0025865_session2_bg</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dbpedia-link</td>
<td>617.1</td>
<td>54.97</td>
</tr>
<tr>
<td>rec-movie lens</td>
<td>427.2</td>
<td>63.57</td>
</tr>
<tr>
<td>soc-TSOPF-RS-b2383-c1</td>
<td>237.7</td>
<td>55.46</td>
</tr>
<tr>
<td>scc_twitter-copen</td>
<td>217.2</td>
<td>95.89</td>
</tr>
<tr>
<td>soc_flickr-und</td>
<td>741.1</td>
<td>329.8</td>
</tr>
<tr>
<td>soc_livejournal_user-groups</td>
<td>2344</td>
<td>133.1</td>
</tr>
<tr>
<td>soc_orkut</td>
<td>613.2</td>
<td>100.2</td>
</tr>
<tr>
<td>soc_orkut_dir</td>
<td>538.9</td>
<td>95.34</td>
</tr>
<tr>
<td>soc_sinaweibo</td>
<td>667.4</td>
<td>90.57</td>
</tr>
<tr>
<td>twitter_npi</td>
<td>413.3</td>
<td>4117</td>
</tr>
<tr>
<td>web_wikipedia_link_it</td>
<td>203.4</td>
<td>190.3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Instance</th>
<th>$\omega_0 (G)$</th>
<th>$w(C_0)$</th>
<th>$r_t$</th>
<th>$r_t'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>aff-digg</td>
<td>3836</td>
<td>2721</td>
<td>0.168</td>
<td>0.380</td>
</tr>
<tr>
<td>aff-orkut-user2groups</td>
<td>971</td>
<td>213</td>
<td>0.849</td>
<td>0.019</td>
</tr>
<tr>
<td>bio-human-gene1</td>
<td>13743</td>
<td>113692</td>
<td>0.283</td>
<td>0.949</td>
</tr>
<tr>
<td>bio-human-gene2</td>
<td>135310</td>
<td>132080</td>
<td>0.353</td>
<td>0.953</td>
</tr>
<tr>
<td>bio-mouse-gene</td>
<td>59952</td>
<td>44472</td>
<td>0.403</td>
<td>0.573</td>
</tr>
<tr>
<td>bn-human-BNU1</td>
<td>20598</td>
<td>16749</td>
<td>0.074</td>
<td>0.517</td>
</tr>
<tr>
<td>0025865_session1_bg</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bn-human-BNU1</td>
<td>19189</td>
<td>8411</td>
<td>0.054</td>
<td>0.463</td>
</tr>
<tr>
<td>0025865_session2_bg</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dbpedia-link</td>
<td>50512</td>
<td>1152</td>
<td>0.186</td>
<td>0.022</td>
</tr>
<tr>
<td>rec-movie lens</td>
<td>3777</td>
<td>2167</td>
<td>0.949</td>
<td>0.381</td>
</tr>
<tr>
<td>soc-TSOPF-RS-b2383-c1</td>
<td>960</td>
<td>415</td>
<td>0.994</td>
<td>0.010</td>
</tr>
<tr>
<td>scc_twitter-copen</td>
<td>58699</td>
<td>57995</td>
<td>0.090</td>
<td>0.918</td>
</tr>
<tr>
<td>soc_flickr-und</td>
<td>10127</td>
<td>6698</td>
<td>0.040</td>
<td>0.292</td>
</tr>
<tr>
<td>soc_livejournal_user-groups</td>
<td>1054</td>
<td>489</td>
<td>0.413</td>
<td>0.038</td>
</tr>
<tr>
<td>soc_orkut</td>
<td>5452</td>
<td>1898</td>
<td>0.756</td>
<td>0.071</td>
</tr>
<tr>
<td>soc_orkut_dir</td>
<td>6147</td>
<td>1170</td>
<td>0.862</td>
<td>0.079</td>
</tr>
<tr>
<td>soc_sinaweibo</td>
<td>4759</td>
<td>834</td>
<td>0.126</td>
<td>0.026</td>
</tr>
<tr>
<td>twitter_npi</td>
<td>13524</td>
<td>9101</td>
<td>0.033</td>
<td>0.192</td>
</tr>
<tr>
<td>web_wikipedia_link_it</td>
<td>89947</td>
<td>89539</td>
<td>0.003</td>
<td>0.987</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Instance</th>
<th>WLMC</th>
<th>WMLC</th>
<th>Cliquer</th>
<th>LSCC+BMS</th>
<th>FastWClq</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1000.9</td>
<td>7341</td>
<td>8471</td>
<td>1165</td>
<td>9072</td>
<td>8552</td>
</tr>
<tr>
<td>C2000.5</td>
<td>2466</td>
<td>2466</td>
<td>2466</td>
<td>2466</td>
<td>2449</td>
</tr>
<tr>
<td>C2000.9</td>
<td>7862</td>
<td>10034</td>
<td>529</td>
<td>10333</td>
<td>9516</td>
</tr>
<tr>
<td>C4000.5</td>
<td>2438</td>
<td>2698</td>
<td>1233</td>
<td>2792</td>
<td>2542</td>
</tr>
<tr>
<td>DSC1000.5</td>
<td>2186*</td>
<td>2186*</td>
<td>2186*</td>
<td>2186*</td>
<td>2186*</td>
</tr>
<tr>
<td>MANN_pa45</td>
<td>34265</td>
<td>34133</td>
<td>1276</td>
<td>34183</td>
<td>34121</td>
</tr>
<tr>
<td>MANN_pa81</td>
<td>111139</td>
<td>111033</td>
<td>195</td>
<td>111094</td>
<td>110481</td>
</tr>
<tr>
<td>hammering10-2</td>
<td>50512*</td>
<td>50512*</td>
<td>50512*</td>
<td>50512*</td>
<td>50512*</td>
</tr>
<tr>
<td>hammering10-4</td>
<td>4812</td>
<td>4614</td>
<td>735</td>
<td>5129</td>
<td>4990</td>
</tr>
<tr>
<td>kellerb</td>
<td>4780</td>
<td>6316</td>
<td>511</td>
<td>7360</td>
<td>5772</td>
</tr>
<tr>
<td>p_bat1000-1</td>
<td>1514*</td>
<td>1514*</td>
<td>1514*</td>
<td>1514*</td>
<td>1514*</td>
</tr>
<tr>
<td>p_bat1000-2</td>
<td>5777*</td>
<td>5777*</td>
<td>5612</td>
<td>5777*</td>
<td>5777*</td>
</tr>
<tr>
<td>p_bat1000-3</td>
<td>8068</td>
<td>7588</td>
<td>2417</td>
<td>8111</td>
<td>7967</td>
</tr>
<tr>
<td>p_bat1500-1</td>
<td>1619*</td>
<td>1619*</td>
<td>1619*</td>
<td>1619*</td>
<td>1619*</td>
</tr>
<tr>
<td>p_bat1500-2</td>
<td>7360</td>
<td>7104</td>
<td>2897</td>
<td>7360</td>
<td>7355</td>
</tr>
<tr>
<td>p_bat1500-3</td>
<td>9846</td>
<td>8449</td>
<td>1497</td>
<td>10278</td>
<td>9875</td>
</tr>
<tr>
<td>san1000</td>
<td>1716*</td>
<td>1716*</td>
<td>929</td>
<td>1716*</td>
<td>1716*</td>
</tr>
</tbody>
</table>

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 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 for all the instances. The two
heuristic algorithms LSCC+BMS and FastWClq cannot find any optimal solution for some graphs within the cutoff time.

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.

<table>
<thead>
<tr>
<th>Group</th>
<th>WLMC # time</th>
<th>MWCLQ # time</th>
<th>Cliquer # time</th>
<th>LSCC+BMS # time</th>
<th>FastWClq # time</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL-500-1000</td>
<td>100 117</td>
<td>100 97.1</td>
<td>20 377</td>
<td>88 101</td>
<td>0 -</td>
</tr>
<tr>
<td>REL-1000-1000</td>
<td>100 2.81</td>
<td>100 2.44</td>
<td>100 5.58</td>
<td>100 4.62</td>
<td>77 103</td>
</tr>
<tr>
<td>REL-1000-500</td>
<td>100 0.11</td>
<td>100 0.12</td>
<td>100 0.11</td>
<td>100 0.23</td>
<td>100 2.08</td>
</tr>
<tr>
<td>REL-1000-1500</td>
<td>100 2.26</td>
<td>100 2.48</td>
<td>100 4.46</td>
<td>100 8.25</td>
<td>78 130</td>
</tr>
<tr>
<td>REL-1500-1500</td>
<td>100 3.24</td>
<td>100 3.71</td>
<td>100 5.14</td>
<td>100 5.09</td>
<td>83 108</td>
</tr>
</tbody>
</table>

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 University. The third author was supported by Mobility Grant PRX16/00215 of the Ministerio de Educaci´on, Cultura y Deporte, the Generalitat de Catalunya grant AGAUR 2014-SGR-118, and the MINECO-FEDER project RASO TIN2015-71799-C2-1-P.

References


