ECAI 2016
G.A. Kaminka et al. (Eds.)
© 2016 The Authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/978-1-61499-672-9-939

Combining Efficient Preprocessing and Incremental MaxSAT Reasoning for MaxClique in Large Graphs

Hua Jiang[1] and **Chu-Min Li**[2] and **Felip Manyà**[3]

Abstract. We describe a new exact algorithm for MaxClique, called LMC (short for Large MaxClique), that is especially suited for large sparse graphs. LMC is competitive because it combines an efficient preprocessing procedure and incremental MaxSAT reasoning in a branch-and-bound scheme. The empirical results show that LMC outperforms existing exact MaxClique algorithms on large sparse graphs from real-world applications.

1 INTRODUCTION

In an undirected graph G=(V, E), where V is the set of vertices and E is the set of edges, a *clique* C is a subset of V such that all its vertices are adjacent to each other. The size of C is its cardinality. The density of G is computed as $2\times|E|/(|V|\times(|V|-1))$. The *Maximum Clique Problem* (MaxClique) is to find a clique of maximum size in G, denoted by $\omega(G)$.

MaxClique is *NP-Hard* [12] and has many practical applications such as fault diagnosis [6], bioinformatics and chemoinformatics [9], coding theory [10], economics [7], and social network analysis [2]. The most deeply studied MaxClique algorithms are exact algorithms based on the branch-and-bound (BnB) scheme [1, 8, 11, 16, 17, 20, 22, 29, 32]. There are also efficient heuristic algorithms such as [5, 13, 23, 24] that find approximate solutions.

In recent years, special attention has been paid to large graphs from real-world applications such as graphs compiled from Internet, social networks, biological networks, collaboration networks and interaction networks. They usually have very low density, contain a high amount of vertices, and have common statistical properties such as small-world property, power-law degree distributions, and clustering [21]. Certainly, cliques are also a valuable property for analyzing such graphs. For example, in biological networks, a clique might be a functional group; in social networks of acquaintance, a clique might identify an organization or a community; and in web networks, a clique might help to find a certain topic.

State-of-the-art exact MaxClique algorithms are effective in solving DIMACS [14] and randomly generated graphs, but unfortunately very few of them are able to solve large graphs from real-world applications. PMC [25] and BBMCSP [27] are exceptions. They were designed to solve large sparse graphs. PMC implements a BnB scheme, and uses approximate graph coloring bounds to prune search and parallelization to speed up the algorithm. BBMCSP is a very recent algorithm for large sparse graphs, which derives from an efficient bitstring encoding and bit-parallel algorithm called BBMCI [28].

Standard MaxSAT reasoning was proposed for MaxClique in [20] and was combined with an incremental upper bound in [16]. Incremental MaxSAT reasoning was proposed in [17] and has shown a superiority over standard MaxSAT reasoning for MaxClique. However, it is very hard to make sophisticated techniques such as incremental MaxSAT reasoning or standard MaxSAT reasoning effective for large graphs, so that it is noted in [27] that sophisticated techniques such as MaxSAT reasoning are not useful for large graphs. In this paper, we present a new exact MaxClique algorithm for large graphs, called LMC (short for Large MaxClique), that combines a novel efficient preprocessing and incremental MaxSAT reasoning. The empirical results on a representative sample of large sparse graphs from real-word applications show that LMC is a fast algorithm for graphs with millions of vertices, and substantially outperforms PMC and BBMCSP, proving that MaxSAT reasoning can be very effective for large graphs.

The paper is organized as follows. Section 2 describes the new algorithm and the techniques it implements. Section 3 reports and analyzes the empirical results. Finally, Section 4 concludes.

2 LMC: A NEW ALGORITHM FOR LARGE SPARSE GRAPHS

We describe algorithm LMC, designed for large sparse graphs, which is composed of two main components: an efficient preprocessing procedure *Initialize* and a main search procedure *SearchMaxClique* employing incremental MaxSAT Reasoning.

2.1 The preprocessing procedure

Preprocessing in MaxClique BnB algorithms is decisive for efficiency, especially for solving large sparse graphs. Preprocessing generally performs the following three tasks:

- Derive a vertex ordering for search;
- Find an initial clique;
- Reduce the input graph; i.e., remove as many vertices that do not belong to any maximum clique as possible.

We define a novel preprocessing procedure called *Initialize*, which performs efficiently all these three tasks at the same time. The starting point is the notion of *core*, which was first used in social network analysis [30]. Let $deg_G(v)$ denote the number of vertices that are adjacent to v in graph $G = (V, E)$, or degree of v in G (G is omitted when it is clear from the context). A subgraph G' induced by $V' \subseteq V$, written as G'=$G(V')$, is a core of order k or a *k-core*

[1] Huazhong University of Sciences and Technology (HUST) & JiangHan University, China, email: jh_hgt@163.com
[2] Corresponding author, MIS, Université de Picardie Jules Verne, France, email: chu-min.li@u-picardie.fr
[3] Artificial Intelligence Research Institute (IIIA, CSIC), Spain, email: felip@iiia.csic.es

iff $deg_{G'}(v) \geq k$ for each $v \in V'$, and G' is the subgraph with the largest number of vertices with this property. The core number of a vertex v, denoted by $k(v)$, is the highest order of a core that contains v. The core number of a graph $G = (V, E)$, denoted by $k(G)$, is the maximum core number among the vertices of G. Refer to the graph in Figure 1. The set $\{v_1, v_2, v_3, v_4, v_5\}$ induces a 2-core, and the set $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ induces a 1-core. The core number of the graph is 2, because the core number of v_6 is 1 and the core number of the other vertices is 2.

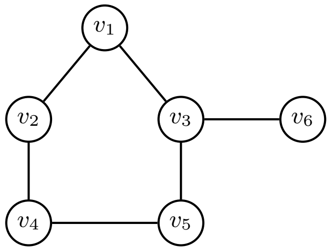

Figure 1. A graph with $k(G)$=2 and $\omega(G)$=2.

The core number of a graph $G = (V, E)$, as well as the core number of each vertex in V, can be efficiently computed in time $O(|E|)$ with an algorithm based on the following property [3, 4]: If we recursively delete all vertices of degree less than k and all edges incident with them in a graph G, the remaining subgraph is the *k-core* of G.

If a graph G has a clique C of size r, then G must have a core of order greater than or equal to $r - 1$, because $deg_{G(C)}(v) = r - 1$ for each $v \in C$. So, $k(G) \geq \omega(G) - 1$; i.e., $\omega(G) \leq k(G) + 1$. Similarly, if a vertex v is in a clique C of size r, then $k(v) \geq r - 1$. In other words, if we want to find a clique of size greater than r, we just need to consider the vertices v such that $k(v) \geq r$. The vertices whose core number is less than r can be discarded, because they cannot be in any clique of size larger than r.

Procedure *Initialize*, showed in Algorithm 1, performs the reduction of the input graph G based on the core number of each vertex of G. It removes all the vertices whose core number is less than a lower bound lb of $\omega(G)$. The procedure should be called by a BnB algorithm searching for a clique of size larger than lb. At the same time, it also derives an initial clique C_0 and determines an initial vertex ordering O_0 for the subsequent search. In the pseudo-code, cur_core denotes the order of the current core, max_core denotes the core number of graph G, and $core\_number[v_i]$ denotes the core number of vertex v_i.

To compute the core number of each vertex of G, procedure *Initialize* first sorts all vertices of V in increasing degree ordering (i.e., $deg(v_1) \leq deg(v_2) \leq \cdots \leq deg(v_n)$), and assigns $deg(v_1)$ to cur_core. G is a core of order cur_core, because $deg(v_i) \geq cur\_core$ for each $v_i \in V$ and G is the largest subgraph with this property. Moreover, cur_core is the core number of the smallest vertex v_1, because it is the greatest order of a core that contains v_1. Afterwards, the procedure considers the set $V \backslash \{v_1\}$: updating the degree of the vertices adjacent to v_1 (line 12), moving them for keeping the increasing degree ordering (line 13). If the degree of v_2 is greater than cur_core, then the subgraph induced by $V \backslash \{v_1\}$ is a new core of order greater than cur_core. In this case, cur_core is updated with the new order. Otherwise, v_2 still belongs to a core of order cur_core. In this way, the core number of all the vertices of G can be computed successively (line 6).

Note that if the set of vertices $\{v_i, v_{i+1}, \ldots, v_{|V|}\}$ is in the increasing ordering of their degree in the subgraph induced by $\{v_i, v_{i+1}, \ldots, v_{|V|}\}$ and the smallest v_i is adjacent to the other vertices (i.e., $deg(v_i) = |V| - i$), then all pairs of vertices in $\{v_i, v_{i+1}, \ldots, v_{|V|}\}$ must be adjacent. So, $\{v_i, v_{i+1}, \ldots, v_{|V|}\}$ forms a clique in this case. As soon as a vertex v_i with such a property is found, the algorithm ends the loop and the core number of all the vertices greater than or equal to v_i is set to cur_core (line 7-10), because they cannot belong to a core of order greater than cur_core. As a result, $\{v_i, v_{i+1}, \ldots, v_{|V|}\}$ is the initial clique C_0 of G (line 14). If $|C_0|$ is greater than lb, lb is set to $|C_0|$. Finally, the vertices with core number less than lb are removed from G, because they cannot be in any clique larger than lb.

Algorithm 1: Initialize(G, lb), a preprocessing for large sparse graphs

```
   Input: $G$=($V$, $E$), a lower bound $lb$ of $\omega(G)$
   Output: an initial clique $C_0$, the core number of $G$, a reduced
           graph $G'$ of $G$, and an initial vertex ordering $O_0$
1  begin
2      Sort $V$ in increasing degree ordering;
3      $cur\_core \leftarrow deg(v_1)$;
4      for $i$:= 1 to $|V|$ do
5          if $deg(v_i) > cur\_core$ then $cur\_core \leftarrow deg(v_i)$;
6          $core\_number[v_i] \leftarrow cur\_core$;
7          if $deg(v_i) = |V| - i$ then
8              for $j$:= $i + 1$ to $|V|$ do
9                  $core\_number[v_j] \leftarrow cur\_core$;
10             break;
11         for each neighbor $v$ of $v_i$ in $\{v_{i+1}, v_{i+2}, \ldots, v_{|V|}\}$ do
12             $deg(v) \leftarrow deg(v) - 1$;
13             Move $v$ and re-index vertices $\{v_{i+1}, v_{i+2}, \ldots, v_{|V|}\}$
               for keeping the increasing degree ordering;
14     $C_0 \leftarrow \{v_i, v_{i+1}, \ldots, v_{|V|}\}$;
15     $max\_core \leftarrow$ the maximum core number in vertices of $V$;
16     if $|C_0| > lb$ then $lb \leftarrow |C_0|$;
17     $G' \leftarrow$ reduced $G$ by removing all vertices with core number
       less than $lb$;
18     $O_0 \leftarrow$ the ordering in which the core number of each vertex
       is computed;
19     return ($C_0$, $max\_core$, $G'$, $O_0$);
```

PMC and BBMCSP also use core numbers in their preprocessing to reduce the input graph. Algorithm 1 differs from the preprocessing of PMC and BBMCSP in that Algorithm 1 performs the following three tasks at the same time: derive the vertex ordering for the subsequent search, find an initial clique, and reduce the graph G by computing the core number of each vertex. However, PMC and BBMCSP perform the three tasks separately. In fact, PMC and BBMCSP use the original algorithm proposed in [4] to compute the core number of each vertex. After computing the core number of a vertex v, it only updates the degree of the vertices adjacent to v and with degree greater than $deg(v)$. In Algorithm 1, the degrees of *all* uncomputed vertices adjacent to v are updated. So, the vertex ordering that Algorithm 1 uses to compute the core numbers (i.e., v_1 is the vertex with the smallest degree in G, v_2 is the vertex with the smallest degree in G after removing v_1, and so on) is exactly the *degeneracy ordering*, which is a typical initial vertex ordering proposed in [8] and used in many BnB MaxClique algorithms. Algorithm 1 returns this ordering as the initial ordering O_0 for the subsequent search. Furthermore, Algorithm 1 naturally derives the initial clique C_0 in line 14, while PMC uses a heuristic [25] to derive C_0 in a separate sub-procedure that can roughly be stated as follows: Let C_0 be the largest clique found so far. For each vertex v in decreasing core number order, if the core number of v is greater than or equal to $|C_0|$, greedily form a clique with v and the neighbors of v. That heuristic has time complexity $O(|E|\cdot\triangle(G))$, where $\triangle(G)$ is the maximum degree in graph G. BBMCSP uses a similar heuristic to derive C_0 [27].

The complexity of Algorithm 1 is dominated by the computation of the core number of vertices. Observe that the cost to compute C_0 and O_0 is negligible. So, although *Initialize* performs more tasks than the original algorithm in [4], the complexity of Algorithm 1 remains $O(|E|)$. Note that the complexity of the preprocessing in PMC and BBMCSP is dominated by the computation of C_0, which is in $O(|E|\cdot\triangle(G))$.

2.2 The BnB algorithm for MaxClique with incremental MaxSAT reasoning

The graph G preprocessed by procedure *Initialize* will be solved by a BnB MaxClique algorithm called *SearchMaxClique*, which is presented in this section. In the first subsection, we describe the branch and bound scheme in *SearchMaxClique*. In the second subsection, we present incremental MaxSAT reasoning for *SearchMaxClique* to reduce the search space.

2.2.1 The BnB scheme in SearchMaxClique

SearchMaxClique is an improved variant of algorithm DoMC [17]. Given a graph G, whose vertices are ordered following a given ordering O, *SearchMaxClique* searches recursively for a clique larger than the current maximum clique C_{max}, combined with the growing clique C.

Algorithm 2: SearchMaxClique(G, C_{max}, C, O), an algorithm for finding a clique of size larger than $|C_{max}|$

```
Input: G=(V, E), the largest clique C_max found so far, the
       current growing clique C, a vertex ordering O
Output: a clique C, if |C|>|C_max|, otherwise, C_max
1  begin
2      if |V|=0 then return C;
3      B ← GetBranches(G, |C_max|-|C|, O);
4      if B=∅ then return C_max;
5      A ← V\B;
6      Let B={b_1, b_2, ..., b_|B|} in the increasing ordering 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          C' ← SearchMaxClique(G(P), C_max, C∪{b_i}, O);
10         if |C'| > |C_max| then C_max ← C' ;
11     return C_max;
```

Algorithm 2 shows the pseudo-code of *SearchMaxClique*. If the set of vertices V is non-empty, it calls function *GetBranches* to partition V into two sets A and B, respecting the vertex ordering O, in such a way that the size of a maximum clique in A is not greater than $|C_{max}| - |C|$, and B=$V\backslash A$=$\{b_1, b_2, \ldots, b_{|B|}\}$ is called the set of branching vertices. If B is empty, the search is pruned and it returns the current largest clique C_{max}. Otherwise, the algorithm searches recursively for a maximum clique, containing $b_i \in B$ and of size greater than $|C_{max}|$, in the subgraphs induced by $\Gamma(b_i)\cap(\{b_{i+1}, b_{i+2}, \ldots, b_{|B|}\}\cup A)$ for $i = |B|, \ldots, 1$, where $\Gamma(b_i)$ denotes the set of vertices adjacent to b_i in G and $b_1 < b_2 < \cdots < b_{|B|}$ w.r.t. O.

Algorithm 3: GetBranches(G, r, O), for an algorithm searching for a maximum clique with more than r vertices in G.

```
Input: G=(V, E), an integer r and a vertex ordering O over V
Output: a set B of branching vertices
1  begin
2      B ← ∅; Π ← ∅; /* Π is a set of Independent Sets (IS) */
3      while V is not empty do
4          v ← the biggest vertex of V w.r.t O;
5          remove v from V;
6          if there is an IS D in Π in which v is not adjacent to any
           vertex then
7              insert v into D;
8          else if |Π|<r then
9              create a new IS D = {v}; Π ← Π ∪ {D};
10         else
11             if There is an IS D_1 in which v has only one
               adjacent vertex u, and u can be inserted into
               another IS D_2 then
12                 insert u into D_2; insert v into D_1;
13             else B ← B ∪ {v};
14     B ← IncMaxSAT(G, O, V\B, B);
15     return B;
```

Function *GetBranches(G, r, O)* is described in Algorithm 3, where r is an integer and O is a vertex ordering. *GetBranches* returns a set B of branching vertices by showing that A=$V\backslash B$ does not contain any clique of size greater than r. The set B is derived by using a greedy sequential coloring process that successively assigns the smallest possible color to each vertex in V w.r.t. ordering O. Note that each color is represented by a natural integer from 1, and that all the vertices with the same color form an independent set (IS) in which no vertex is adjacent to another. The vertices that cannot be assigned a color smaller than or equal to r form the set B. Since vertices in A=$V\backslash B$ can be colored using r colors, they cannot form a clique of size larger than r.

Since the greater the cardinality of B, the greater the remaining search space to be explored, *GetBranches* reduces B using procedure *Re-NUMBER* [32]. It works as follows: when the coloring process fails to color a vertex v with a color less than or equal to r, it checks whether there exists a vertex u and two ISs D_1 and D_2, such that u is the only vertex adjacent to v in D_1 but there is no vertex adjacent to u in D_2. In this case, u is moved to D_2 and v is inserted into D_1, obtaining this way one additional vertex in A and one less in B.

Finally, *GetBranches* further reduces B by applying incremental MaxSAT reasoning [17], which is described in the next subsection.

2.2.2 Incremental MaxSAT reasoning

We first explain the rationale to use MaxSAT reasoning in BnB MaxClique algorithms. Then, we give basic notions of MaxSAT and describe standard MaxSAT reasoning for MaxClique. Finally, we describe incremental MaxSAT reasoning.

BnB MaxClique algorithms compute upper bounds (UBs) of $\omega(G)$ to prune search [22, 31, 32, 20, 19, 29], using approximate algorithms such as greedy sequential coloring to derive good quality UBs with low overhead. Given a graph G = (V, E), greedy sequential coloring successively assigns the smallest possible color (from 1) to each vertex v in a predefined ordering, and the computed UB of $\omega(G)$ is the greatest color r needed to color G. Let D_i={v| $v \in V$ and v is assigned color i} for i=1,..., r. D_i is an IS, and the vertex coloring process partitions V into r ISs. In Figure 1, assuming that a coloring process colors the vertices in V in the ordering v_1, v_2, v_3, v_4, v_5, v_6, then three ISs D_1={v_1, v_4, v_6}, D_2={v_2, v_3} and D_3={v_5} are obtained.

The coloring process has a low time complexity $O(n^2)$. Nevertheless, the derived UB may not be tight enough; e.g., in Figure 1, UB=3 and $\omega(G)$=2. A subset of q ISs is said to be *conflicting* if the q ISs cannot form a clique of size q. Recent approaches [20, 19] use *standard MaxSAT reasoning* to improve the coloring-based UB by detecting disjoint conflicting subsets of ISs, after encoding MaxClique to MaxSAT. They apply the following proposition [20]: Let G be a graph that can be partitioned into r ISs. If the r ISs can be partitioned into c disjoint conflicting subsets of ISs, then $\omega(G) \leq r - c$.

Recall that a literal is a propositional variable x or its negation $\bar{x}$, and a clause is a disjunction of literals. A clause is unit if it contains exactly one literal. A partial MaxSAT (PMaxSAT) instance is a multiset of clauses in which some clauses are declared to be hard and the others are declared to be soft. Given a PMaxSAT instance, the PMaxSAT problem is to find an assignment that satisfies all hard clauses and the maximum number of soft clauses [18].

After G is partitioned into ISs, MaxClique can be reduced to PMaxSAT as follows [20]: Given a graph $G = (V, E)$, we define a propositional variable x_i for each $v_i \in V$ with the intended meaning that x_i is true iff v_i belongs to the maximum clique C_{max}, and derive the PMaxSAT instance ϕ that contains (i) a hard clause $\bar{x}_i \vee \bar{x}_j$ for each pair of non-adjacent vertices v_i and v_j, stating that v_i and v_j cannot be both in C_{max}, and (ii) a soft clause for each IS in the partition, which is the disjunction of the variables associated to the vertices in the IS. Notice that at most one vertex in each IS can be in C_{max}. So, at most one variable can be assigned true in each soft clause. An assignment that satisfies all hard clauses and the maximum number of soft clauses in ϕ identifies a maximum clique in G (containing one vertex per satisfied soft clause).

Nevertheless, the purpose of encoding G as a PMaxSAT instance ϕ in [20] is not to solve ϕ with a MaxSAT solver, but to detect conflicts in ϕ with an approximate PMaxSAT algorithm to improve the coloring-based UB of $\omega(G)$.

Example 1 (from [20]) The vertices of the graph G in Figure 1 can be partitioned into three ISs: {v_1, v_4, v_6}, {v_2, v_3}, {v_5}. The PMaxSAT encoding ϕ is formed by the hard clauses: {$\bar{x}_1 \vee \bar{x}_4$, $\bar{x}_1 \vee \bar{x}_5$, $\bar{x}_1 \vee \bar{x}_6$, $\bar{x}_2 \vee \bar{x}_3$, $\bar{x}_2 \vee \bar{x}_5$, $\bar{x}_2 \vee \bar{x}_6$, $\bar{x}_3 \vee \bar{x}_4$, $\bar{x}_4 \vee \bar{x}_6$, $\bar{x}_5 \vee \bar{x}_6$}, and the soft clauses: {$x_1 \vee x_4 \vee x_6$, $x_2 \vee x_3$, x_5}. Standard MaxSAT Reasoning detects a conflict as follows. Assume that x_5 is true (i.e., v_5 is in C_{max}). Literal $\bar{x}_5$ must be removed from the hard clauses $\bar{x}_1 \vee \bar{x}_5$, $\bar{x}_2 \vee \bar{x}_5$, and $\bar{x}_5 \vee \bar{x}_6$, resulting in three hard unit clauses: $\bar{x}_1$, $\bar{x}_2$, and $\bar{x}_6$. The satisfaction of these three unit clauses removes x_1, x_2 and x_6 from the soft clauses $x_1 \vee x_4 \vee x_6$ and $x_2 \vee x_3$, and results in two soft unit clauses: x_4 and x_3. The satisfaction of these two new soft unit clauses makes the hard clause $\bar{x}_3 \vee \bar{x}_4$ falsified. Hence, the three soft clauses $x_1 \vee x_4 \vee x_6$, $x_2 \vee x_3$ and x_5 cannot be satisfied simultaneously, meaning that the three corresponding ISs cannot form a clique of size 3. Consequently, the coloring-based UB of $\omega(G)$ is improved from 3 to 2.

The improvement of UB depends on the number of disjoint conflicts detected. Standard MaxSAT reasoning has no impact if the improved UB is still greater than the size of the largest clique found so far. It is easy to see that the greater the average cardinality of ISs, the harder to detect conflicts. This fact might explain why standard MaxSAT reasoning in sparse graphs is not as useful as in medium and dense graphs, because the ISs in sparse graphs are generally very large.

Algorithm 4: IncMaxSAT(G, O, A, B), incremental MaxSAT reasoning to reduce the set of branching vertices B

```
Input: G=(V, E), V is ordered w.r.t. O and is partitioned into
       two subsets: A and B, and A is partitioned into ISs
Output: a set of branching vertices B
1  begin
2      $\phi \leftarrow$ the PMaxSAT instance: hard clauses encoded from $G$
       and soft clause according to the IS partition of $A$;
3      while $B$ is not empty do
4          $v \leftarrow$ the biggest vertex in $B$ w.r.t. $O$;
5          Add the soft unit clause $\{v\}$ in $\phi$;
6          if a conflict can be detected in $\phi$ then
7              Let $c_1, c_2, \ldots, c_p$ are the conflicting soft clauses;
8              Remove $c_1, c_2, \ldots, c_p$ from $\phi$;
9              Add the soft clauses $c_1 \vee z_1, c_2 \vee z_2, \ldots, c_p \vee z_p$ to $\phi$;
10             Add the constraint $z_1 + z_2 + \cdots + z_p = 1$ to $\phi$;
11             Remove $v$ from $B$;
12         else break;
13     return $B$;
```

To remedy that drawback of standard MaxSAT reasoning, *incremental MaxSAT reasoning* was proposed in [17], resulting in two efficient algorithms: DoMC and SoMC. These algorithms partition the set of vertices V into two sets, A and B, in such a way that the vertices in A are colored with $|C_{max}|$ colors, and B=$V \setminus A$={b_1, b_2, ..., $b_{|B|}$} is the set of branching vertices. If B is empty, the search is pruned. Otherwise, the algorithms get the PMaxSAT instance ϕ: the hard clauses encode the non-adjacent vertices of G, and there is a soft clause for each IS of the partition of A. The MaxSAT encoding is implicit. The propositional variables are directly represented by the vertices, the soft clauses are represented by the corresponding ISs, and the hard clauses are represented using the adjacency matrix. In addition, each vertex is associated with the list of the non-adjacent vertices. In this way, no extra space is needed, and the cost of the encoding is negligible once A and B are obtained.

Then, incremental MaxSAT reasoning successively adds to ϕ the highest vertex b_i of B as a soft unit clause {b_i} for i=|B| to 1, respecting the ordering O. If a new conflict is detected in ϕ after adding {b_i}, then b_i is removed from B and added to A. Let c_1, c_2, ..., c_p be the soft clauses involved in the detected conflict. Note that at least one of these soft clauses is falsified by each truth assignment that satisfies all the hard clauses. These soft clauses are weakened before detecting the next conflict: A fresh propositional variable z_i is added to soft clause c_i for i=1 to p, and a hard constraint $z_1 + z_2 + \cdots + z_p = 1$ is added to require that exactly one of these variables is assigned true. The weakened soft clauses can then be used to detect further conflicts. The constraint $z_1 + z_2 + \cdots + z_p = 1$ is treated as follows: if one variable z_i is assigned true, the other variables are assigned false. Adding the fresh variables and the hard constraint allows one to satisfy exactly one soft clause that was previously falsified by each assignment.

If $b_{|B|}$, $b_{|B|-1}$, ..., and b_i are removed from B and added to A, and a conflict is detected for each one of these vertices, then the set $A \cup$ {$b_{|B|}$, $b_{|B|-1}$, ..., b_i} cannot form a clique of size greater than

$|C_{max}|$. To see this, note that the PMaxSAT instance ϕ contains now $|C_{max}|$+$|B|$-i+1 soft clauses and, for each truth assignment satisfying all the hard clauses, $|B|$-i+1 soft clauses have to be satisfied because of the fresh variables. So, at most $|C_{max}|$ soft clauses can be satisfied by the vertices.

If a conflict cannot be detected when adding a soft unit clause $\{b_i\}$ to ϕ, the reduced set B=$\{b_1, b_2, \ldots, b_i\}$ is returned. Algorithm 4 shows the pseudo-code of incremental MaxSAT reasoning.

Example 2 (adapted from [17]). Refer to the graph G in Figure 1. Assume that $|C_{max}|$=2, and a coloring process partitions the graph into A=$\{D_1, D_2\}$ and B=$\{v_5, v_6\}$, where D_1=$\{v_1, v_4\}$ and D_2=$\{v_2, v_3\}$. *IncMaxSAT* encodes the graph into a PMaxSAT instance ϕ by directly treating the vertices as Boolean variables, the ISs D_1 and D_2 as soft clauses, and the non-adjacency relations between vertices as hard clauses. Then, it adds a new soft unit clause $\{v_5\}$ to ϕ, and proves that $\{D_1, D_2, \{v_5\}\}$ cannot form a clique of size 3 as follows. If v_5 is in the clique (i.e., v_5 is assigned true), then v_1 and v_2 cannot be in the clique (i.e., v_1 and v_2 should be assigned false), because they are non-adjacent to v_5. So, the only remaining vertices v_4 in D_1 and v_3 in D_2 should be in the clique, which is impossible because v_4 and v_3 are non-adjacent (if so, the hard clause $\bar{v}_3 \vee \bar{v}_4$ would be falsified). So the three soft clauses D_1, D_2 and $\{v_5\}$ are conflicting.

Then, *IncMaxSAT* adds a fresh variable z_1 (z_2, z_3) to D_1 (D_2, $\{v_5\}$) together with the hard constraint z_1+z_2+z_3=1, before adding the soft unit clause $\{v_6\}$ to ϕ. It proves that the three soft clauses D_1=$\{v_1, v_4, z_1\}$, $\{v_5, z_3\}$ and $\{v_6\}$ are conflicting as follows. Assume that v_6 is true. Then, v_1, v_4 and v_5 should be false, because they are not adjacent to v_6. However, z_1 and z_3 cannot both be true because of the hard constraint z_1+z_2+z_3=1. A fresh variable z_4 (z_5, z_6) is then added to D_1 ($\{v_5, z_3\}$ and $\{v_6\}$) together with the hard constraint z_4+z_5+z_6=1.

The two conflicts are clearly disjoint. Recall that ϕ contains now four soft clauses. Given a truth assignment satisfying all the hard clauses, at most two soft clauses can be satisfied due to the original variables v_i ($1 \le i \le 6$), and the other two soft clauses are satisfied due to the fresh variables z_i ($1 \le i \le 6$). This shows that $A \cup \{v_5, v_6\}$ cannot form a clique of size greater than 2.

The advantage of incremental MaxSAT reasoning over standard MaxSAT reasoning for MaxClique is that if it eliminates all the branching vertices, then the search is pruned; otherwise, the set of branching vertices is generally significantly reduced. We note that although large real-world graphs are usually sparse, they might contain cliques with hundreds of vertices. So, BnB MaxClique algorithms will develop wider and deeper search trees for such graphs. The reduction of the set of branching vertices B by incremental MaxSAT reasoning has a dramatic impact on performance when hard large sparse graphs are solved, as we will see in Section 3.

2.3 Algorithm *LMC* for large sparse graphs

Algorithm 5 describes LMC, which is especially suited for large sparse graphs. Roughly speaking, given a graph G, LMC calls procedure *Initialize* to preprocess both G and the first level subgraphs in the search tree, and then calls the search procedure *SearchMaxClique* to recursively search for a maximum clique in the reduced subgraphs.

LMC first calls $Initialize(G, 0)$ (the initial lb of $\omega(G)$ is 0) to derive an initial clique C_0, the core number of G and of each vertex, a reduced subgraph G' and an initial ordering O_0. If the size of C_0 is $k(G)+1$, then C_0 is a maximum clique of G and is returned (line 3), because $k(G)+1$ is an UB of $\omega(G)$. Otherwise, LMC 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, where vertices follow the initial ordering O_0. For each first level subgraph $G(P)$ of the search tree, LMC calls $Initialize(G(P), |C_{max}| - 1)$ to compute an initial clique C'_0 of $G(P)$, the core number of $G(P)$ and of each vertex in $G(P)$, a subgraph G'' of $G(P)$ obtained by removing all the vertices whose core number is less than $|C_{max}| - 1$, and a vertex ordering O'_0. Finally, the search procedure *SearchMaxClique* is called to recursively search for a clique containing v_i, of size greater than $|C_{max}|$, in the subgraph G''. Observe that the size of a maximum clique in $G(P)$ is at most $k(G(P))$+1. When $k(G(P))$+1<$|C_{max}|$, a clique containing v_i of size greater than $|C_{max}|$ cannot be found from G'' and the search in G'' is pruned.

Algorithm 5: LMC(G), a BnB algorithm for MaxClique in large sparse graphs

```
Input: $G$=($V$, $E$)
Output: a maximum clique $C_{max}$ of $G$
1  begin
2      ($C_0$, $k(G)$, $G'$, $O_0$) $\leftarrow Initialize(G, 0)$;
3      if $|C_0| = k(G) + 1$ then return $C_0$ ;
4      $C_{max} \leftarrow C_0$;
5      $V' \leftarrow$ the vertex set of $G'$;
6      Order $V'$ w.r.t the initial ordering $O_0$;
7      for $i$:= $|V'|$ to 1 do
8          $P \leftarrow \Gamma(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_{|V'|}\}$;
9          ($C'_0$, $k(G(P))$, $G''$, $O'_0$) $\leftarrow$
10                 $Initialize(G(P), |C_{max}| - 1)$;
11         if $|C'_0| \ge |C_{max}|$ then $C_{max} \leftarrow C'_0 \cup \{v_i\}$;
12         if $k(G(P)) + 1 \ge |C_{max}|$ then
13             Construct the adjacency matrix for $G''$;
14             $C' \leftarrow SearchMaxClique(G'', C_{max}, \{v_i\}, O'_0)$;
15             if $|C'| > |C_{max}|$ then $C_{max} \leftarrow C'$ ;
16     return $C_{max}$
```

LMC also calls *Initialize* for the first level subgraphs. The rationale is: (i) the vertex ordering computed by *Initialize* is *degeneracy ordering*; re-ordering the vertices in the subgraphs near the root of the search tree was proven to be beneficial for BnB MaxClique algorithms [15]. (ii) Since G is large, the first level subgraphs may still contain a lot of vertices. With a growing lower bound $|C_{max}|$ of $\omega(G)$, the first level subgraphs can be further reduced, which is beneficial for speeding up the search in *SearchMaxClique*. We note that the first level subgraphs are also preprocessed in BBMCSP.

Maintaining an adjacency matrix for large sparse graphs has a costly space complexity. LMC does not construct a global adjacency matrix for the input graph G. When the search in the first level subgraphs is necessary, the adjacency matrix for them is constructed dynamically to serve *SearchMaxClique* (line 13). Since G is sparse, the number of vertices in the first level subgraphs should be substantially reduced by the preprocessing in line 9 and line 10. So, constructing an adjacency matrix for the reduced first level subgraphs is feasible and beneficial. In the implementation, we use bit-sets to store the adjacency matrix.

3 EMPIRICAL INVESTIGATION

We empirically evaluated LMC, and compared it with PMC and BBMCSP, which are, to our best knowledge, the two most efficient

Table 1. The graphs tested in the experiments, excluding those graphs whose $init$ and $search$ times are below 10s for all solvers.

Graph	$\|V\|$	$\|E\|$	Graph	$\|V\|$	$\|E\|$	Graph	$\|V\|$	$\|E\|$
adaptive	6815744	13624320	inf-great-britain_osm	7733822	8156517	soc-ljournal-2008	5363186	49514271
aff-digg	872622	22501700	inf-road-usa	23947347	28854312	soc-orkut	2997166	106349209
aff-flickr-user-groups	395979	8537703	packing-500x100 x100-b050	2145852	17488243	soc-orkut-dir	3072441	117185083
aff-orkut-user2groups	8730857	327036486	rec-amazon-ratings	2146057	5743132	soc-pokec	1632803	22301964
bio-human-gene1	22283	12323680	rec-dating	168792	17351416	soc-sinaweibo	58655849	261321033
bio-human-gene2	14340	9027024	rec-eachmovie	74424	2811458	soc-twitter-higgs	456631	12508442
bio-mouse-gene	45101	14461095	rec-epinions	755761	13396042	soc-wiki-conflict	118100	2027871
bn-human-BNU_1_0 025864_session_1-bg	1827218	143158339	rec-libimseti-dir	220970	17233144	soc-youtube-growth	3223589	9376594
bn-human-BNU_1_0 025864_session_2-bg	1827241	133727516	rec-movielens	71567	9991339	socfb-A-anon	3097165	23667394
bn-human-BNU_1_0 025865_session_1-bg	1398408	42296922	rgg_n_2_23_s0	8388608	63501393	socfb-B-anon	2937612	20959854
bn-human-BNU_1_0 025865_session_2-bg	1717207	22855526	rgg_n_2_24_s0	16777216	132557200	socfb-konect	59216214	92522012
channel-500x100 x100-b050	4802000	42681372	sc-TSOPF-RS -b2383-c1	38120	16115324	socfb-uci-uni	58790782	92208195
dbpedia-link	11621692	78621046	sc-ldoor	952203	20770807	tech-as-skitter	1694616	11094209
delaunay_n22	4194304	12582869	sc-rel9	5921786	23667162	tech-ip	2250498	21643497
delaunay_n23	8388608	25165784	soc-BlogCatalog	88784	2093195	tech-p2p	5792297	147829887
delaunay_n24	16777216	50331601	soc-FourSquare	639014	3214986	twitter_mpi	9862152	99940317
friendster	8658744	45671471	soc-LiveJournal1	4847571	42851237	web-ClueWeb09-50m	428136613	446534058
hugebubbles-00020	21198119	28857767	soc-buzznet	101163	2763066	web-baidu-baike	2141300	17014946
hugetrace-00000	4588484	6879133	soc-catster	149700	5448197	web-indochina-2004-all	7414865	150984819
hugetrace-00010	12057441	18082179	soc-digg	770799	5907132	web-it-2004-all	41291318	1027474947
hugetrace-00020	16002413	23998813	soc-dogster	426820	8543549	web-uk-2002-all	18520343	261787258
ia-enron-email-dynamic	87273	297456	soc-flickr-und	1715255	15555041	web-wiki-ch-internal	1930275	8956902
ia-wiki-Talk-dir	2394385	4659565	soc-flixster	2523386	7918801	web-wikipedia-growth	1870709	36532531
ia-wiki-user-edits-page	2104544	5572584	soc-friendster	65608366	1806067135	web-wikipedia_link_en	27154756	31024475
inf-europe_osm	50912018	54054660	soc-livejournal -user-groups	7489073	112305407	web-wikipedia_link_it	2936413	86754664
inf-germany_osm	11548845	12369181	soc-livejournal07	5204176	48709773			

exact MaxClique algorithms for large sparse graphs. LMC was implemented in C and compiled using GNU gcc -O3. The algorithms are also called solvers when they are used to solve MaxClique instances. The experiments were performed on an Intel Xeon CPU X5460@3.16GHz under Linux with 32GB of memory.

We next describe the compared solvers and the tested graphs, and then discuss and analyze the experimental results.

3.1 The compared solvers and the tested graphs

The source code of PMC [25] is publicly available at https://www.cs.purdue.edu/homes/dgleich/codes/maxcliques/. We compiled it using the provided Makefile and ran it with ./pmc -f G -a 0 to solve G. We used the Linux binary executable of BBMCSP [27] available at http://venus.elai.upm.es/logs/results_sparse/bin/.

The measures considered for each tested graph are the following:

The size of initial clique C_0 (ω_0). PMC and BBMCSP use a dedicated heuristic to compute C_0, while LMC derives C_0 in procedure *Initialize* as a by-product of computing core numbers.

The time for preprocessing ($init$). For PMC and BBMCSP, it includes the time of the k-core analysis, finding C_0, and reducing the graph. For LMC, it is the time needed by procedure *Initialize* to preprocess the input graph at the root of the search tree. It does not include the preprocessing of the first level subgraphs.

The time for search ($search$). The runtime after finishing the preprocessing. For LMC, it includes both the time of search and preprocessing of first level subgraphs. The cut-off time was set to 5 hours.

We considered 170 real-world graphs from the Network Data Repository [26] available at http://networkrepository.com, including the 90 graphs reported to evaluate BBMCSP in [27]. The number of vertices ranges from 4K to 400M. We exclude the graphs whose $init$ and $search$ times are below 10s for all the solvers and report results for the remaining 77 graphs, providing a clearer comparison. Table 1 shows the number of vertices ($|V|$) and number of edges ($|E|$) of these 77 graphs.

3.2 Comparison of LMC with PMC and BBMCSP

Table 2 shows the experimental results, where $k(G)$+1 is the UB of $\omega(G)$ derived by the k-core analysis. When the size of an initial clique (ω_0) is $k(G)$+1, the search time is 0 in LMC and BBMCSP because no search is performed. PMC does not return $search = 0$ in some cases for some unknown reason. The best times are in bold.

LMC solves all graphs in at most 1120s, while PMC (BBMCSP) cannot solve 12 (6) instances in 5h. In addition, BBMCSP runs out of memory on two graphs. Furthermore, LMC is almost always faster than BBMCSP and PMC. For example, for *bio-mouse-gene*, the total time ($init$+$search$) of LMC is 147s, which is 4.4 and 35.5 times faster than PMC (643s) and BBMCSP (5220s); for *soc-sinaweibo*, the total time of LMC is 80s, which is 59 and 40 times faster than PMC (4742s) and BBMCSP (3206s), respectively.

Observe that the $init$ times of LMC are significantly below the ones of PMC and BBMCSP due to the overhead of the dedicated heuristic for finding the initial clique in PMC and BBMCSP. Surprisingly, although the method for finding an initial clique in LMC is very simple, it finds larger initial cliques than the ones found by PMC and BBMC for 14 and 18 instances, respectively. For example, the initial clique found by LMC for *tech-p2p* is of size 172, while the initial clique found by PMC and BBMCSP is of size 155 and 153, respectively.

Although the initial clique found by LMC is smaller than the one found by PMC and BBMCSP in many cases, the search after the preprocessing of LMC is almost always faster than PMC and BBMCSP. For example, the search time of LMC for *rec-epinions* is 123 and 20 times faster than PMC and BBMCSP, respectively; and LMC finishes the search in 150s for *twitter_mpi*, whereas PMC and BBMCSP cannot terminate in 5h. For these two graphs, the initial cliques found by LMC (resp. 2 and 79) are smaller than the ones found by PMC (resp. 7 and 113) and BBMCSP (resp. 7 and 121). This also happens on

Table 2. Runtimes in seconds of LMC, PMC and BBMCSP to solve the large graphs in Table 1. For each graph, the cpu time limit is 5h, ω_0 is the size of the initial clique found by the solvers, $init$ denotes the time for preprocessing, and $search$ denotes the time for search after the preprocessing.

Graph	$k(G)$ +1	ω	LMC			PMC			BBMCSP		
			ω_0	init	search	ω_0	init	search	ω_0	init	search
adaptive	3	2	2	**1.74**	**0.01**	2	4.25	6.09	2	13.42	1.44
aff-digg	646	32	29	**1.98**	**175.6**	24	62.38	>5h	25	49.01	1397
aff-flickr-user-groups	187	14	12	**0.85**	**3.62**	10	9.43	53.06	10	15.79	22.42
aff-orkut-user2groups	472	6	2	**83.06**	**436.5**	6	932.4	>5h	5	2410	3727
bio-human-gene1	2048	1335	1328	**0.65**	**1047**	1272	90.48	>5h	1268	16.55	>5h
bio-human-gene2	1903	1300	1290	**0.48**	**178.4**	1241	66.54	>5h	1229	7.77	>5h
bio-mouse-gene	1046	561	435	**1.29**	**145.8**	525	33.06	610.4	520	24.35	5196
bn-human-BNU_1_0025864_session_1-bg	1210	294	222	**14.72**	**642.8**	271	177.8	>5h	276	277.3	>5h
bn-human-BNU_1_0025864_session_2-bg	1088	271	199	**13.91**	**544.1**	271	154.9	1595	271	252.4	912.8
bn-human-BNU_1_0025865_session_1-bg	912	196	159	**3.90**	**192.9**	172	56.16	12611	186	84.43	>5h
bn-human-BNU_1_0025865_session_2-bg	582	201	83	**2.18**	40.91	201	20.73	120.9	201	33.76	**12.64**
channel-500x100x100-b050	10	4	4	**10.19**	**2.10**	4	18.96	32.37	4	33.44	11.41
dbpedia-link	140	33	10	**17.62**	**16.07**	30	4050	3774	32	243.4	118.1
delaunay_n22	5	4	3	**2.69**	**0.12**	4	6.21	13.54	4	15.75	2.51
delaunay_n23	5	4	3	**5.60**	**0.25**	4	12.46	25.70	4	32.26	5.13
delaunay_n24	5	4	3	**11.55**	**0.50**	4	24.76	32.82	4	67.06	10.71
friendster	52	37	17	**8.26**	**1.42**	37	18.71	24.36	37	67.91	5.56
hugebubbles-00020	3	2	2	**15.78**	**0.01**	2	23.47	41.27	2	62.42	6.78
hugetrace-00000	3	2	2	**2.24**	**0.01**	2	3.52	4.91	2	10.04	1.06
hugetrace-00010	3	2	2	**6.35**	**0.01**	2	9.60	16.51	2	30.21	2.99
hugetrace-00020	3	2	2	**10.14**	**0.01**	2	16.19	27.23	2	47.16	4.30
ia-enron-email-dynamic	54	33	24	**0.02**	**0.02**	28	0.48	16.03	30	0.15	**0.02**
ia-wiki-Talk-dir	132	26	25	**0.35**	**0.37**	22	4.18	8.01	16	12.56	0.89
ia-wiki-user-edits-page	67	15	13	**0.27**	**0.10**	14	689.9	>5h	11	46.60	0.34
inf-europe_osm	4	4	3	9.34	0.01	4	**7.93**	**0.00**	4	12.83	**0.00**
inf-germany_osm	4	3	3	2.13	**0.01**	3	**2.06**	2.57	3	16.43	**0.01**
inf-great-britain_osm	4	3	3	**1.32**	**0.01**	3	1.47	1.78	3	11.10	**0.01**
inf-road-usa	4	4	3	7.93	0.01	3	**0.01**	5.96	4	10.13	**0.00**
packing-500x100x100-b050	10	4	4	**3.17**	**0.94**	4	8.16	11.96	4	15.21	5.75
rec-amazon-ratings	30	5	4	**1.15**	**0.18**	4	3.39	5.32	4	10.40	2.24
rec-dating	261	13	8	**1.76**	**12.16**	11	11.89	161.4	9	35.32	55.74
rec-eachmovie	221	12	11	**0.24**	**0.81**	11	9.68	49.98	10	2.87	1.51
rec-epinions	149	8	2	**1.69**	**2.50**	7	169.0	308.8	7	44.00	51.87
rec-libimseti-dir	274	14	13	**1.79**	**10.47**	12	14.97	159.3	11	38.46	45.53
rec-movielens	532	29	24	**0.96**	**17.72**	23	18.08	541.8	20	14.05	127.0
rgg_n_2_23_s0	21	21	21	**9.16**	**0.00**	21	11.48	**0.00**	21	15.92	**0.00**
rgg_n_2_24_s0	21	21	21	**19.43**	**0.00**	21	25.13	**0.00**	21	33.80	**0.00**
sc-TSOPF-RS-b2383-c1	656	7	3	**0.84**	6.82	7	3.28	162.7	6	3.25	**2.65**
sc-ldoor	35	21	21	**1.50**	2.46	21	10.56	9.85	21	5.83	**1.84**
sc-rel9	5	4	3	**2.03**	**0.19**	4	6.53	21.14	4	28.76	4.56
soc-BlogCatalog	222	45	41	**0.16**	**1.81**	39	2.67	14.36	37	2.12	4.02
soc-FourSquare	64	30	25	**0.19**	**0.22**	29	36.82	36.53	27	7.09	0.39
soc-LiveJournal1	373	321	320	**5.30**	0.10	314	8.26	5.71	316	53.89	**0.03**
soc-buzznet	154	31	28	**0.22**	**1.15**	25	4.41	17.56	23	3.52	2.37
soc-catster	420	81	56	**0.32**	4.78	80	8.08	16631	58	6.20	**1.35**
soc-digg	237	50	18	**0.57**	2.56	46	2.37	13.32	42	9.35	**2.43**
soc-dogster	249	44	39	**0.61**	**2.65**	40	10.95	18.42	33	13.94	4.50
soc-flickr-und	569	98	74	**1.49**	**24.32**	77	11.50	1027	68	24.24	155.3
soc-flixster	69	31	30	**0.60**	**0.18**	29	1.46	1.84	29	10.28	0.91
soc-friendster	305	129	14	**726.7**	**393.6**	129	1253	7458		out of memory	
soc-livejournal-user-groups	117	9	4	**19.83**	**42.48**	8	7360	>5h	8	1351	2064
soc-livejournal07	375	358	358	**6.32**	0.05	358	7.21	1.35	356	62.03	**0.01**
soc-ljournal-2008	426	400	389	**5.50**	0.04	400	9.16	2.50	400	47.18	**0.01**
soc-orkut	231	47	17	**23.39**	**29.25**	43	72.84	268.5	46	190.8	85.72
soc-orkut-dir	254	51	14	**26.01**	**34.08**	48	79.94	291.3	50	209.2	96.67
soc-pokec	48	29	15	**4.28**	**1.90**	29	8.30	8.36	29	34.53	6.68
soc-sinaweibo	194	44	8	**53.14**	**26.83**	37	3493	1249	41	3012	194.8
soc-twitter-higgs	126	71	21	**1.35**	**1.71**	71	15.13	28.28	70	29.11	4.43
soc-wiki-conflict	146	25	22	**0.17**	**0.46**	21	7.48	62.82	22	1.75	0.77
soc-youtube-growth	89	20	18	**1.44**	**0.39**	17	51.63	33.97	18	32.41	2.01
socfb-A-anon	75	25	23	**3.75**	**5.02**	23	12.43	22.21	24	40.41	15.48
socfb-B-anon	64	24	11	**3.64**	**4.55**	24	10.44	19.41	24	35.46	14.10
socfb-konect	17	6	6	**17.94**	**0.15**	6	26.20	20.26	6	131.8	1.72
socfb-uci-uni	17	6	6	**20.01**	**0.16**	6	30.28	20.69	6	164.5	1.75
tech-as-skitter	112	67	57	**0.87**	**0.10**	66	1.55	0.87	50	11.09	0.14
tech-ip	254	4	3	**2.22**	**7.47**	3	67.79	>5h	4	103.9	24.76
tech-p2p	854	178	172	**26.21**	**208.8**	155	229.8	>5h	153	564.2	>5h
twitter_mpi	678	131	79	**15.68**	**149.5**	113	1456	>5h	121	537.8	>5h
web-ClueWeb09-50m	189	56	41	**130.9**	**2.40**	55	411.8	258.2		out of memory	
web-baidu-baike	79	31	12	**2.84**	**1.11**	31	38.13	30.62	31	46.77	8.46
web-indochina-2004-all	6870	6848	6848	**5.61**	82.38	6848	3621	162.5	6848	86.62	**0.85**
web-it-2004-all	3225	3222	3222	**47.20**	2.99	3222	682.1	>5h	3222	451.8	**0.45**
web-uk-2002-all	944	944	944	**16.40**	**0.00**	944	35.30	9.71	944	21.29	**0.00**
web-wiki-ch-internal	121	33	13	**1.17**	**0.52**	32	13.67	10.53	33	14.90	2.17
web-wikipedia-growth	207	31	15	**6.26**	**6.41**	31	354.9	273.5	31	131.9	52.22
web-wikipedia_link_en	378	44	4	**6.99**	**2.09**	43	24.79	93.24	44	57.70	13.80
web-wikipedia_link_it	895	870	869	**5.67**	0.25	870	152.1	>5h	869	85.28	**0.06**

other graphs such as *dbpedia-link*, *friendster*, *soc-livejournal-user-groups* and *soc-orkut-dir*. This fact provides evidence that incremental MaxSAT reasoning is effective in pruning search in large sparse graphs, even with a smaller initial clique.

Overall, LMC is clearly superior to PMC and BBMCSP. These results show that the combination of the novel preprocessing and incremental MaxSAT reasoning in the proposed BnB scheme is competitive to find maximum cliques in large sparse graphs, which is analyzed in details in the next subsection.

3.3 The impact of preprocessing and incremental MaxSAT reasoning in LMC

To analyze the impact of preprocessing in LMC, we report in Table 3 the density of the graphs whose search time is more than 5s in Table 2, before and after the preprocessing at the root of the search tree (see line 2 in Algorithm 5), and the ratio of the number of vertices of the reduced graph to the number of vertices of the original graph, as well as the mean density of a subgraph $G(P)$ and of its reduced graph G'' (see line 8 and line 9 in Algorithm 5), and the mean ratio of the number of vertices of G'' to the number of vertices of $G(P)$. We can see that most graphs are already significantly reduced by preprocessing at the root of the search tree, which considerably increases their density. The preprocessing in the first level of the search tree further reduces the graphs and substantially increases the density of G'', which is really solved by the BnB algorithm *SearchMaxClique* (line 14 of Algorithm 5). While an IS in the original graph G with very low density is usually huge, an IS in G'' is much smaller, which explains the effectiveness of incremental MaxSAT reasoning when solving G'' in LMC, because conflicts among small ISs can be easily detected to efficiently reduce the set B of branching vertices.

Table 3. Impact of the preprocessing in LMC. The columns d and d' are respectively the density of the original graph G and of its reduced graph G' in the root of the search tree, rt the ratio of the number of vertices of G' to the number of vertices of G; $\overline{d}_P$ and $\overline{d''}$ the mean density of the subgraph $G(P)$ and of the reduced subgraph G'' in the first level of the tree, and rt'' the ratio of the number of vertices of G'' to the number of vertices of $G(P)$. The entries marked with '-' mean that all vertices of the first level subgraphs have been removed by the preprocessing.

Graph	Original Graph			First Level Subgraphs		
	d	rt	d'	$\overline{d}_P$	$\overline{rt''}$	$\overline{d''}$
aff-digg	0.000059	0.16	0.002066	0.302	0.32	0.35
aff-orkut-user2groups	0.000009	0.78	0.000014	0.005	0.01	0.18
bio-human-gene1	0.049641	0.20	0.636241	0.968	0.79	0.98
bio-human-gene2	0.087802	0.27	0.693864	0.973	0.80	0.98
bio-mouse-gene	0.014219	0.38	0.080813	0.780	0.28	0.93
bn-human-BNU_1_0 025864_session_1-bg	0.000086	0.13	0.003767	0.591	0.15	0.70
bn-human-BNU_1_0 025864_session_2-bg	0.000080	0.14	0.003416	0.587	0.17	0.69
bn-human-BNU_1_0 025865_session_1-bg	0.000043	0.06	0.009783	0.524	0.29	0.66
bn-human-BNU_1_0 025865_session_2-bg	0.000016	0.05	0.005656	0.461	0.22	0.65
dbpedia-link	0.000001	0.18	0.000028	0.147	0.01	0.52
rec-dating	0.001218	0.81	0.001869	0.056	0.10	0.28
rec-libimseti-dir	0.000706	0.79	0.001128	0.044	0.04	0.28
rec-movielens	0.003902	0.91	0.004667	0.255	0.27	0.35
sc-TSOPF-RS-b2383-c1	0.022181	1.00	0.022181	0.004	0.00	-
soc-flickr-und	0.000011	0.03	0.007760	0.349	0.18	0.62
soc-friendster	0.000001	0.42	0.000004	0.041	0.01	0.33
soc-livejournal -user-groups	0.000004	0.42	0.000022	0.044	0.01	0.37
soc-orkut-dir	0.000025	0.82	0.000036	0.185	0.03	0.40
soc-orkut	0.000024	0.75	0.000039	0.191	0.02	0.44
soc-sinaweibo	0.000001	0.12	0.000006	0.055	0.01	0.61
socfb-A-anon	0.000005	0.13	0.000191	0.163	0.01	0.77
tech-ip	0.000009	0.19	0.000206	0.002	0.01	0.15
tech-p2p	0.000009	0.03	0.003944	0.236	0.09	0.67
twitter_mpi	0.000002	0.03	0.001355	0.241	0.10	0.64
web-indochina -2004-all	0.000005	0.01	0.999713	0.999	0.89	0.99
web-wikipedia -growth	0.000021	0.49	0.000071	0.172	0.02	0.43

We now show the individual impact of preprocessing the first level subgraphs and of incremental MaxSAT reasoning in LMC by comparing it with the following solvers:

LMC\prep1. It is LMC without preprocessing the first level subgraphs; i.e., line 9 and line 10 that call procedure *Initialize* are removed in Algorithm 5.

LMC\MaxSAT. It is LMC without incremental MaxSAT reasoning; i.e., line 14 in the GetBranches function (Algorithm 3) is removed.

Table 4 shows the search tree size and the search time of LMC, LMC\prep1 and LMC\MaxSAT on the graphs of Table 3. With preprocessing and incremental MaxSAT reasoning, the search tree size of LMC is always the smallest. The search time of LMC is comparable with that of LMC\prep1 and LMC\MaxSAT on easy graphs. However, LMC is substantially faster than LMC\prep1 and LMC\MaxSAT on hard graphs. In particular, the search time of LMC is smaller than 1047s for all the graphs, while LMC\prep1 fails to solve 4 graphs and LMC\MaxSAT fails to solve 2 graphs within 5h. In addition, LMC is 14 and 4 times faster than LMC\MaxSAT for *bio-human-gene1* and *tech-p2p*, respectively.

Table 4. Search tree sizes in thousands and search times in seconds (s) of LMC, LMC\prep1 and LMC\MaxSAT. The cpu time limit is 5h.

Graph	LMC		LMC\prep1		LMC\MaxSAT	
	tree	search	tree	search	tree	search
aff-digg	**5602**	**175.6**	7055	181.5	13553	202.9
aff-orkut-user2groups	**4173**	436.5	6831	**316.6**	**4173**	457.3
bio-human-gene1	**11.19**	**1047**	12.53	1081	59.46	14618
bio-human-gene2	**14.16**	178.4	15.45	**122.8**	35.85	392.4
bio-mouse-gene	**76.92**	**145.8**	78.85	153.2	611.1	703.1
bn-human-BNU_1_0 025864_session_1-bg	**210.1**	**642.8**	-	>5h	-	>5h
bn-human-BNU_1_0 025864_session_2-bg	**215.4**	**544.1**	-	>5h	-	>5h
bn-human-BNU_1_0 025865_session_1-bg	**104.3**	**192.9**	-	>5h	7490	1184
bn-human-BNU_1_0 025865_session_2-bg	**53.70**	40.91	105.7	60.11	96.31	**40.17**
dbpedia-link	**684.6**	16.07	686.7	**13.30**	685.0	16.12
rec-dating	**138.3**	12.16	142.8	**9.87**	**138.3**	12.21
rec-libimseti-dir	**173.3**	10.47	174.7	**9.19**	**173.3**	10.43
rec-movielens	**120.8**	17.72	144.3	**10.51**	160.1	17.81
sc-TSOPF-RS-b2383-c1	**34.25**	**6.82**	34.26	11.87	**34.25**	6.83
soc-flickr-und	**81.01**	**24.32**	307.0	61.49	639.7	39.22
soc-friendster	**2308**	393.6	2318	412.3	**2308**	**388.8**
soc-livejournal-user-groups	**2403**	42.48	**2403**	**31.32**	**2403**	42.88
soc-orkut-dir	**714.0**	34.08	731.5	**27.81**	724.3	34.14
soc-orkut	**745.5**	29.25	751.6	**23.87**	754.5	29.51
soc-sinaweibo	**712.7**	**26.83**	714.4	30.50	713.5	26.93
socfb-A-anon	**355.2**	5.02	**355.2**	**3.76**	**355.2**	4.99
tech-ip	**79.29**	**7.47**	**79.29**	9.70	**79.29**	7.48
tech-p2p	**235.1**	**208.8**	1116	746.5	2922	831.0
twitter_mpi	**323.1**	**149.6**	-	>5h	2911	249.7
web-indochina-2004-all	**0.14**	82.38	**0.14**	**24.36**	**0.14**	82.45
web-wikipedia-growth	**358.4**	6.41	359.6	**5.18**	360.7	6.35

4 CONCLUSIONS

Clique is a valuable property for analyzing real-world large sparse graphs. We have proposed a new exact algorithm, called LMC, to find maximum cliques in large sparse graphs that combines a novel preprocessing and incremental MaxSAT reasoning in a BnB scheme. The preprocessing procedure *Initialize* performs effectively three tasks at the same time with a very low overhead: derive a vertex ordering, reduce the graph, and compute an initial clique. LMC also applies preprocessing to the subgraphs in the first level of the search tree, so that the underlying algorithm *SearchMaxClique* can work with a better vertex ordering and a more reduced graph when searching these subgraphs with incremental MaxSAT reasoning.

Our new algorithm can solve all tested graphs efficiently and shows a performance that is superior over PMC and BBMCSP, refuting the opinion in the literature that sophisticated techniques such as MaxSAT reasoning are not useful for large sparse graphs. The empirical analysis suggests that the performance of incremental MaxSAT reasoning in LMC comes from the fact that the proposed preprocessing considerably increases the density of the graphs to be solved.

ACKNOWLEDGEMENTS

This work is supported by NSFC Grant No. 61370184, No. 61272014, and No. 61472147. 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

[1] L. Babel and G. Tinhofer, 'A branch and bound algorithm for the maximum clique problem', *Methods and Models of Operations Research*, **34**(3), 207–217, (1990).

[2] B. Balasundaram, S. Butenko, and I.V. Hicks, 'Clique relaxations in social network analysis: The maximum k-plex problem', *Operations Research*, **59**(1), 133–142, (2011).

[3] V. Batagelj and A. Mrvar, 'Pajek: a program for large network analysis', *Connections*, **21**(2), 47–57, (1998).

[4] V. Batagelj and M. Zaversnik, 'An o(m) algorithm for cores decomposition of networks', *Eprint Arxiv Cs*, **1**(6), 34–37, (2003).

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

[6] P. Berman and A. Pelc, 'Distributed probabilistic fault diagnosis for multiprocessor systems', in *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, pp. 340–346, (1990).

[7] V. Boginski, S. Butenko, and P.M. Pardalos, 'Mining market data: A network approach.', *Computers & Operations Research*, **33**(11), 3171–3184, (2006).

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

[9] M.S. Dawn, B. Earl, and Joel S.S., 'Optimal protein structure alignment using maximum cliques', *Operations Research*, **53**(3), 389–402, (2005).

[10] T. Etzion and P.R.J. Ostergard, 'Greedy and heuristic algorithms for codes and colorings', *IEEE Transactions on Information Theory*, **44**(1), 382–388, (1998).

[11] T. Fahle, 'Simple and fast: Improving a branch-and-bound algorithm for maximum clique', *Esa*, **2461**, 47–86, (2002).

[12] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[13] A. Grosso, M. Locatelli, and W. Pullan, 'Simple ingredients leading to very efficient heuristics for the maximum clique problem', *Journal of Heuristics*, **14**(6), 587–612, (2008).

[14] D.J. Johnson and M.A. Trick, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*, American Mathematical Society, 1996.

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

[16] C.M. Li, Z.W. Fang, and K. Xu, 'Combining maxsat reasoning and incremental upper bound for the maximum clique problem', in *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pp. 939–946, (2013).

[17] C.M. Li, H. Jiang, and R.C. Xu, 'Incremental maxsat reasoning to reduce branches in a branch-and-bound algorithm for maxclique', *Lecture Notes in Computer Science*, **8994**, 268–274, (2015).

[18] C.M. Li and F. Manyà, 'MaxSAT, hard and soft constraints', in *Handbook of Satisfiability*, eds., Armin Biere, Hans van Maaren, and Toby Walsh, 613–631, IOS Press, (2009).

[19] C.M. Li and Z. Quan, 'Combining graph structure exploitation and propositional reasoning for the maximum clique problem', in *Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, pp. 344–351, (2010).

[20] C.M. Li and Z. Quan, 'An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem', in *Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, Usa, July*, pp. 128–133, (2010).

[21] M.E.J. Newman, 'The structure and function of complex networks', in *SIAM Rev*, pp. 167–256, (2003).

[22] P.R.J. Ostergard, 'A fast algorithm for the maximum clique problem', in *Discrete Appl. Math*, pp. 197–207, (2002).

[23] W. Pullan and H.H. Hoos, 'Dynamic local search for the maximum clique problem', *J. Artif. Int. Res.*, **25**(1), 159–185, (2006).

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

[25] R.A. Rossi, D.F. Gleich, A.H. Gebremedhin, and M.M.A. Patwary, 'Parallel maximum clique algorithms with applications to network analysis and storage', *Eprint Arxiv*, (2013).

[26] R.A. Rossi and K.A. Nesreen, 'The network data repository with interactive graph analytics and visualization', in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, (2015).

[27] P.S. Segundo, L. Alvaro, and P.M. Pardalos, 'A new exact maximum clique algorithm for large and massive sparse graphs', *Computers & Operations Research*, **66**, 81–94, (2016).

[28] P.S. Segundo, F. Matia, D. Rodriguez-Losada, and M. Hernando, 'An improved bit parallel exact maximum clique algorithm', *Optimization Letters*, **7**(3), 467–479, (2011).

[29] P.S. Segundo, D. Rodríguez-Losada, and A. Jiménez, 'An exact bit-parallel algorithm for the maximum clique problem', *Computers & Operations Research*, **38**(2), 571–581, (2011).

[30] S.B. Seidman, 'Network structure and minimum degree', *Social Networks*, **5**(3), 269–287, (1983).

[31] E. Tomita and T. Kameda, 'An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments', *Journal of Global Optimization*, **37**(1), 95–111, (2007).

[32] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki, 'A simple and faster branch-and-bound algorithm for finding a maximum clique', in *WALCOM: Algorithms and Computation, 4th International Workshop, WALCOM 2010*, pp. 191–203, (2010).