Combining MaxSAT Reasoning and Incremental Upper Bound for the Maximum Clique Problem

Chu-Min LI[1] Zhiwen Fang[1,2] Ke Xu[2]

[1]*MIS, Universit de Picardie Jules Verne*
Amiens, France,
chu-min.li@u-picardie.fr, zhiwenf@gmail.com

[2]*State Key Lab. Of Software Development Environment*
Beihang University, China
kexu@nlsde.buaa.edu.cn

***Abstract*—Recently, MaxSAT reasoning has been shown to be powerful in computing upper bounds for the cardinality of a maximum clique of a graph. However, existing upper bounds based on MaxSAT reasoning have two drawbacks: (1) at every node of the search tree, MaxSAT reasoning has to be performed from scratch to compute an upper bound and is time-consuming; (2) due to the NP-hardness of the MaxSAT problem, MaxSAT reasoning generally cannot be complete at a node of a search tree, and may not give an upper bound tight enough for pruning search space. In this paper, we propose an incremental upper bound and combine it with MaxSAT reasoning to remedy the two drawbacks. The new approach is used to develop an efficient branch-and-bound algorithm for MaxClique, called IncMaxCLQ. We conduct experiments to show the complementarity of the incremental upper bound and MaxSAT reasoning and to compare IncMaxCLQ with several state-of-the-art algorithms for MaxClique.**

***Keywords*-MaxClique; MaxSAT; Incremental Upper Bound**

I. Introduction

Consider an undirected graph G=(V, E), where V is a set of n vertices $\{v_1, v_2, ..., v_n\}$ and E is a set of m edges. An edge (v_i, v_j) $\in E$ if and only if v_i and v_j are adjacent. A clique of G is a subset C of V such that every two vertices in C are adjacent. The maximum clique problem (MaxClique for short) consists in finding a clique of G of the largest cardinality. An independent set of G is a subset I of V such that no two vertices in I are adjacent.

Maxclique is a very important NP-hard combinatorial problem, because it appears in many real-world applications such as bioinformatics (see e.g. [1]) and fault diagnosis (see e.g. [3]). A huge amount of effort has been devoted to solve it. Two main types of algorithms (also called solvers) are developed for Maxclique: heuristic methods, e.g., [18], [8], [19], [2], and exact algorithms including branch-and-bound algorithms, e.g., [21], [22], [23], [24], [11], [12], [10], [16], [4], [7], [20]. Recently, a computational study is performed on exact algorithms for MaxClique [17]. In this paper, we focus on exact algorithms for Maxclique based on the branch-and-bound scheme, which prove the optimality of the solutions they find.

The cardinality of a maximum clique of a graph G is usually denoted by $\omega(G)$. Most branch-and-bound algorithms for MaxClique compute an upper bound for $\omega(G)$ by partitioning G into independent sets: If a graph can be partitioned into r independent sets, then $\omega(G) \leq r$, since every independent set contains at most one vertex in the maximum clique. This upper bound may not be very tight especially when G is not perfect. Recently, MaxSAT reasoning has been shown to be very powerful in improving the upper bound based on independent sets [11], [12]. However, existing upper bounds based on MaxSAT reasoning have two drawbacks: (1) MaxSAT reasoning has to be performed from scratch in every node of the search tree and is time-consuming; (2) due to the NP-hardness of the MaxSAT problem, MaxSAT reasoning generally cannot be complete at a node of a search tree, so that the improved upper bound may still not be tight enough for pruning search space.

In this paper, we propose an incremental upper bound and combine it with MaxSAT reasoning to remedy the two drawbacks. Differently from an upper bound computed using MaxSAT reasoning from scratch in quadratic time, the incremental upper bound is derived in linear time from the bounds computed for subgraphs of G. Besides being faster, incremental upper bound is complementary to MaxSAT reasoning because it may be tighter in practice. We then develop a new MaxClique solver called IncMaxCLQ using the new approach.

This paper presents the new approach and is organized as follows. Section II presents preliminaries. Section III presents the incremental bound and its combining with MaxSAT reasoning in IncMaxCLQ. The incremental upper bound is based on a total vertex ordering in G presented in Section IV. Section V discusses related works. Section VI analyzes experimental results that show the complementarity of incremental bound and MaxSAT reasoning in MaxClique solving, and compare IncMaxCLQ with several state-of-the-art solvers. Section VII concludes the paper.

II. Preliminaries

Let V' be a subset of V, the subgraph of G induced by V' is defined as $G(V')$=(V', E'), where E'=$\{(v_i, v_j) \in E \mid v_i, v_j \in V'\}$. The set of adjacent (or neighbor) vertices of a vertex v in G is denoted by $\Gamma(v)$=$\{v'|(v, v')$

$\in E\}$. The cardinality $|\Gamma(v)|$ of $\Gamma(v)$ is called the degree of v. $G\backslash v$ denotes the subgraph induced by $V\backslash\{v\}$, i.e. $G\backslash v$=$G(V\backslash\{v\})$. The density of a graph of n vertices and m edges is $2m/(n(n-1))$.

G can have several maximum cliques. In order to search for a maximum clique of G, a branch-and-bound algorithm usually selects a vertex v and divides all cliques of G into two sets: the set of cliques containing v and the set of cliques not containing v. Then the algorithm recursively searches for a clique of the largest cardinality in each set. Let LB be the cardinality of the largest clique found so far in G, and UB be an upper bound of the largest cardinality of any clique containing v. If UB$\leq$LB, search in cliques containing v can be pruned, since no clique larger than LB can be found. Most branch-and-bound algorithms for MaxClique use the following property to compute the upper bound UB.

Property 1. If a graph G can be partitioned into r independent sets, then $\omega(G)$ is bounded by r.

Computing the smallest r is NP-hard since it is equivalent to the graph coloring problem. The heuristic method used to computing r in [22] requires $O(n^2)$ time, where n is the number of vertices in G. The method proposed in [24] gives better r but requires $O(n^3)$ time. We denote an upper bound computed using Property 1 by UB$_{IndSet}$ (for UB based on Independent Set).

Unfortunately, UB$_{IndSet}$ may not be tight, especially when the graph is imperfect. A recent approach proposed in [11], [12] uses MaxSAT reasoning to improve UB$_{IndSet}$. To be self contained, we adapt the following example from [11] to illustrate how MaxSAT reasoning works to improve UB$_{IndSet}$.

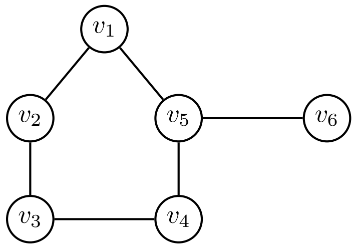

Figure 1. A simple graph

An optimal partition of the graph G in Figure 1 is $\{v_1, v_3, v_6\}$, $\{v_2, v_4\}$, $\{v_5\}$. So, UB$_{IndSet}$=3. MaxSAT reasoning performs as follows: assume that each of the three independent sets contributes a vertex to the maximum clique under construction, then v_5 is in the clique, excluding v_2 and v_3 from the clique because they are not adjacent to v_5, so the only remaining v_4 in the second set should be in the clique, excluding v_1 and v_6 from the clique. Therefore, the first independent set cannot contribute any vertex to the clique. This contradiction shows that the three independent sets cannot all contribute a vertex in the maximum clique. So, $\omega(G)$ is at most 2 (i.e., UB=2), instead of 3.

The set of independent sets such as $\{\{v_1, v_4, v_6\}, \{v_2, v_3\}, \{v_5\}\}$ is said *conflicting*, since at least one independent set in the set cannot contribute a vertex to the maximum clique. In general, we have:

Property 2 ([11], [12]). Suppose that a graph G can be partitioned into r independent sets, and that these independent sets can be partitioned into k disjoint conflicting sets, then $\omega(G)$ is bounded by r-k.

MaxSAT reasoning in [11], [12] essentially uses unit propagation and failed literal detection to detect disjoint conflicting sets s_1,s_2, ..., s_k of independent sets. The upper bound is then improved by k. This reasoning is obviously incomplete and requires $O(n^2)$ time. We denote an upper bound computed using Property 2 by UB$_{MaxSAT}$. Refer to [11], [12] for more details.

Besides the application in MaxClique solving, UB$_{MaxSAT}$ is also shown to be powerful in [13], [14] for MinSAT (Minimum Satisfiability) solving.

III. Incremental Upper Bound for MaxClique and Solver IncMaxCLQ

Given a graph G=(V, E) and a fixed total ordering in V such that v_1<v_2<v_3<...<v_n, we associate an upper bound vertexUB$_G[v_i]$ with every vertex v_i for the size of the largest clique containing v_i in the subgraph of G induced by $\{v_i, v_{i+1}, ..., v_n\}$. Observe that for any vertex v, the value of vertexUB$_G[v]$ depends only on vertices larger than v in the vertex ordering. Refer to the graph in Figure 2, let the vertex ordering be: v_1<v_2<v_3<v_4<v_5<v_6, vertexUB$_G[v_2]\geq$4, vertexUB$_G[v_4]\geq$2, and vertexUB$_G[v_5]\geq$1.

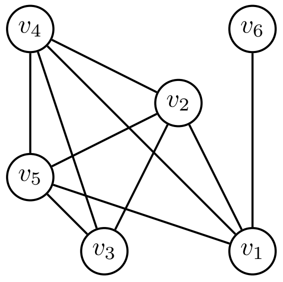

Figure 2. A graph to illustrate incremental upper bound

We propose to use the vertexUB array in a branch-and-bound MaxClique solver based on the following property:

Property 3. Let v be the smallest vertex in G, and $G(\Gamma(v))$ be subgraph of G induced by the neighbors of v. Then $\omega(G(\Gamma(v)))\leq$Max$_{u\in\Gamma(v)}$(vertexUB$_G[u]$).

In other words, when v is the smallest vertex in G, 1+Max$_{u \in \Gamma(v)}$(vertexUB$_G[u]$) is an upper bound for the maximum clique containing v in G (which is a maximum clique in $G(\Gamma(v))$ plus v). The bound is incremental because it is derived in linear time from vertexUB in subgraphs of G specified by a vertex ordering. We denote the incremental upper bound by ***IncUB***. IncUB and vertexUB are tightly related and are new in this paper.

Observe that IncUB requires a fixed vertex ordering to be computed, while UB$_{IndSet}$ and UB$_{MaxSAT}$ presented in the Preliminaries section do not. Given a vertex ordering in G, all the three upper bounds for the maximum clique containing the smallest vertex v in G can be computed. It is easy to see that UB$_{MaxSAT}$ is at least as tight as UB$_{IndSet}$, but that UB$_{MaxSAT}$ is more time-consuming. For small graphs (e.g. with fewer than 100 vertices), UB$_{MaxSAT}$ is very tight, because it is easy to detect conflicting sets of independent sets using MaxSAT reasoning for small graphs. However, for large and complex graphs, our experimental results show that UB$_{MaxSAT}$ and IncUB are complementary, in the sense that sometimes UB$_{MaxSAT}$<IncUB and sometimes IncUB<UB$_{MaxSAT}$, because Property 3 may capture better the structures of the graph than the incomplete MaxSAT reasoning from scratch.

We propose now a new branch-and-bound MaxClique algorithm, called IncMaxCLQ and sketched in Algorithm 1. IncMaxCLQ uses a global array vertexUB, where G is omitted for simplicity because it is clear from context. At the beginning, vertexUB[v] is initialized for each v in G using IncUB (Property 3), according to a fixed vertex ordering (see the next section). Then when branching on v, vertexUB[v] is improved to

min(vertexUB[v], IncUB, UB$_{IndSet}$, UB$_{MaxSAT}$)

which combines MaxSAT reasoning and the incremental upper bound. In practice, the improvement of vertexUB[v] is stopped as soon as its value allows to prune search.

When IncMaxCLQ is called for a subgraph G' of G, it uses the same array vertexUB for G', meaning that vertexUB$_{G'}[v]$ is initialized to the value of vertexUB$_G[v]$ for each v in G'. This initialization is based on:

Property 4. Let G' be a subgraph of G with the same vertex ordering as G. Then $\omega(G'(\Gamma(v)))$+1≤vertexUB$_G[v]$ for any vertex v in G'.

Given G, a fixed vertex ordering, a clique C under construction, and the largest clique C_{max} found so far (both C and C_{max} being initialized to $\emptyset$), IncMaxCLQ first searches a largest clique not containing v in line 4. An induction easily shows that this recursive call of IncMaxCLQ also improves vertexUB for each vertex in $G \backslash v$. Then in line 6, an upper bound for the largest clique containing v is computed to see if search should be pruned in line 7. If not, IncMaxCLQ is recursively called to search for the largest clique containing v in line 9. Observe that this recursive call can change vertexUB[u] for any vertex u in the subgraph. The old value of vertexUB[u] is restored upon backtracking. In line 11, vertexUB[v] is improved by the result of the recursive calling of line 9.

We illustrate the behavior of vertexUB in pruning search using graph G in Figure 2. Just for simplicity in this illustration, we assume that UB$_{MaxSAT}$ is always equal to the number of vertices in G. Table I shows an execution of IncMaxCLQ for this graph using call N^o 1, which makes the recursive call N^o 2 in line 4 (for the subgraph excluding v_1) and the recursive call N^o 3 in line 9 (for the subgraph induced by neighbors of v_1). G, C and C_{max} are showed for each call. We also give the vertexUB value of each vertex obtained using IncUB in each call below each vertex. Observe that call N^o 3 will make 4 further recursive calls, one for each vertex, which are all pruned in line 7, because the vertexUB value of these vertices computed using IncUB in line 6 is smaller than or equal to 3 (=$|C_{max}|$-1), meaning that a clique larger than C_{max} cannot be found.

Algorithm 1: IncMaxCLQ(G, C, C_{max}), an incremental branch and bound algorithm for Maxclique

```
Input: Graph $G$=($V$, $E$), clique $C$ under construction,
       and the largest clique $C_{max}$ found so far
Output: $C \cup C'$, where $C'$ is a maximum clique of $G$,
        if $|C \cup C'|>|C_{max}|$, $C_{max}$ otherwise
1  begin
2      if $|V|$=0 then return $C$;
3      $v \leftarrow$ the smallest vertex of $G$;
4      $C_1 \leftarrow$ IncMaxCLQ($G \backslash v$, $C$, $C_{max}$);
5      if $|C_1| > |C_{max}|$ then $C_{max} \leftarrow C_1$ ;
6      vertexUB[$v$] $\leftarrow$ min(vertexUB[$v$], IncUB,
       UB$_{IndSet}$, UB$_{MaxSAT}$);
7      if $|C_{max}| \geq$ vertexUB[$v$]+$|C|$ then return $C_{max}$;
8      foreach vertex $u$ in $G(\Gamma(v))$, save vertexUB[$u$];
9      $C_2 \leftarrow$ IncMaxCLQ($G(\Gamma(v))$, $C \cup \{v\}$, $C_{max}$);
10     foreach $u$ in $G(\Gamma(v))$, restore the saved
       vertexUB[$u$];
11     vertexUB[$v$] $\leftarrow$ min(vertexUB[$v$], $|C_2|$-$|C|$);
12     if $|C_1| \geq |C_2|$ then return $C_1$; else return $C_2$;
```

N^o	G	C	C_{max}	MaxClique returned
1	v_1, v_2, v_3, v_4, v_5, v_6 5, 4, 3, 2, 1, 1	$\emptyset$	$\emptyset$	v_2, v_3, v_4, v_5
2	v_2, v_3, v_4, v_5, v_6 4, 3, 2, 1, 1	$\emptyset$	$\emptyset$	v_2, v_3, v_4, v_5
3	v_2, v_4, v_5, v_6 3, 2, 1, 1	$\{v_1\}$	v_2, v_3, v_4, v_5	Pruned

Table I
THREE CALLS OF INCMAXCLQ TO SOLVE THE GRAPH IN FIGURE 2

IV. Vertex Ordering in IncMaxCLQ

The ordering for branching in an exact algorithm is a very important issue in solving NP-hard problems such as SAT, CSP, graph coloring and MaxClique. In this paper, vertex ordering should try to minimize the initial vertexUB value for each vertex. Note that these initial values can be inherited by subgraphs. Refer to Figure 1, if vertices are ordered as $v_1<v_2<v_4<v_5<v_3<v_6$, vertexUB[v_1,v_2,v_3,v_4,v_5,v_6] would be initialized to {6, 5, 4, 3, 2, 1} using IncUB. However, if vertices are ordered as $v_5<v_2<v_4<v_1<v_3<v_6$, vertexUB[v_5,v_2,v_4,v_1,v_3,v_6] would be initialized to {3, 2, 2, 1, 1, 1} using IncUB. The second ordering is thus much better than the first one, because it captures better the graph structure.

Observe that in the second ordering, the three last vertices $\{v_1,v_3,v_6\}$ constitute a maximum independent set of the graph, allowing to maximize the number of vertices having the vertexUB value 1, and the two preceding vertices v_2 and v_4 constitute the second maximum independent set of the graph (after removing the first one), maximizing the number of vertices having the vertexUB value 2. Based on this observation, we propose the *MaxIndSet vertex ordering*: let S_1, S_2, ..., S_r be successive maximum independent sets partitioning G, $v_i<v_j$ iff $v_i \in S_a$ and $v_j \in S_b$ such that $a>b$ or $a=b$ and degree(v_i)<degree(v_j).

When every S_a is large, IncUB can be small for a subgraph. However, the MaxIndSet vertex ordering has two drawbacks, making it non-suitable for some graphs:

- In order to compute S_1 in G=(V, E), one should search for a maximum clique S_1 in the complementary graph $\overline{G}$=(V, $\overline{E}$) ((v_i,v_j)$\in\overline{E}$ iff (v_i,v_j)$\notin E$). If G is dense, $\overline{G}$ is sparse and searching for S_1 is generally easy in $\overline{G}$. However, when G is sparse, $\overline{G}$ is dense and searching for a maximum clique in $\overline{G}$ may be much harder than in G. So the MaxIndSet vertex ordering cannot be applied to sparse graphs. In this paper, a graph G is considered to be dense, if its density $\geq$0.7. In this case, the runtime for searching for S_1, S_2, ..., S_r in $\overline{G}$ is negligible in most cases using IncMaxCLQ.
- When G is dense, we can easily obtain the successive maximum independent sets S_1, S_2, ..., S_r partitioning G. However, if G is not regularly structured, there will be at least 2 independent sets of cardinality 1 in the partition. Such a partition is considered *irregular*. The advantage of a large independent set is that when a vertex is selected to be added into a clique, all other vertices in the set are eliminated. Independent sets of cardinality 1 do not allow this advantage. We do not use the MaxIndSet vertex ordering in this case.

When MaxIndSet vertex ordering is not used, we use the classical ordering for MaxClique proposed in [4]: $v_1<v_2<v_3<...<v_n$ such that v_1 is the vertex with the smallest degree, v_2 is the vertex with the smallest degree in G after removing v_1, and so on. This ordering is called *degeneracy ordering* in [6]. Its advantage is that for any v_i, the number of neighbors of v_i among v_{i+1},...,v_n is bounded by the degeneracy of G, which is the smallest value d such that every nonempty subgraph of G contains a vertex of degree at most d [15]. Obviously $\omega(G)\leq d$+1.

Algorithm 2: VertexOrdering(G)

```
Input: A graph $G$=($V$, $E$)
Output: An ordering of $V$
1 begin
2     sort the vertices of $G$ in the degeneracy ordering $v_1<v_2<...<v_n$;
3     if $G$ is not dense, then return the degeneracy ordering;
4     else partition $G$ into independent sets by searching for successive maximum cliques in $\overline{G}$ using IncMaxCLQ with the inverse degeneracy ordering $v_n<v_{n-1}<...<v_1$;
5     if the partition is irregular
6     then return the degeneracy ordering $v_1<v_2<...<v_n$;
7     else return the MaxIndSet vertex ordering;
```

Since the number of neighbors of v_i among $\{v_{i+1}, ..., v_n\}$ is tightly bounded in the degeneracy ordering, IncUB for a maximum clique containing v_i computed using Property 3 is expected to be small with high probability.

Algorithm 2 summarizes the vertex ordering in G used in IncMaxCLQ. Automatically switching between the new MaxIndSet ordering and the degeneracy ordering is another contribution of this paper.

V. Related Works

Most branch-and-bound algorithms for MaxClique use UB$_{IndSet}$ (Property 1) to prune search. The difference among these algorithms lies in how to obtain the partition of the graph G into independent sets. Cliquer [16] partitions G by constructing an independent set at a time: as long as there are vertices that can be added into the set (i.e. non-adjacent to all existing vertices in the set), such a vertex with the largest degree is added into the set and removed from G.

In MCQ [22], MCR [23], MaxCliquedyn [10], and MaxCLQ [11], [12], vertices are inserted into an independent set in a predetermined ordering. Suppose that the current independent sets are S_1, S_2, ..., S_r (in this order, r is 0 at the beginning of the partition process), the current first vertex v is inserted into the first S_i such that v is non-adjacent to all vertices already in S_i. If such a S_i does not exist, a new independent set S_{r+1} is opened and v is inserted here.

Another important component of a branch-and-bound algorithm for MaxClique is its vertex ordering heuristic for branching. MCR, MaxCliquedyn, and MaxCLQ all use the following vertex ordering: the independent sets are sorted in the ordering in which they are created. If the largest clique found so far in G is of cardinality LB, and G can be partitioned into r independent sets S_1, S_2, ..., S_r (created in this order), the algorithms successively branch on vertices in independent sets S_r, S_{r-1}, S_{r-2}, ... , S_{LB+1} (in this order). Note that the algorithms need not branch on vertices in S_1, S_2, ..., and S_{LB}.

The vertex ordering for branching in MCR, MaxCliquedyn, and MaxCLQ is dynamic, because the current graph is partitioned dynamically at every node of the search tree to compute the upper bound for the maximum clique and to obtain a vertex ordering. In Cliquer, the vertex ordering is statically fixed after partitioning the input graph into independent sets in a preprocessing: S_1, S_2, ..., S_r. Vertices are ordered such that vertices in S_i come before vertices in S_j if $i > j$, which is similar to the MaxIndSet ordering used in IncMaxCLQ, except that in Cliquer S_i ($1 \leq i \leq r$) is created as described at the beginning of this section, and is generally not a maximum independent set. Moreover, Cliquer uses this vertex ordering to solve all graphs, while IncMaxCLQ automatically switches between the degeneracy ordering and the MaxIndSet ordering depending on whether or not the graph is regularly structured.

Finally, we note that while vertex ordering in previous MaxClique solvers can be static or dynamic, the vertex ordering in IncMaxCLQ should be static to make subgraphs of G able to inherit the vertexUB information from G.

VI. **Experimental Analysis**

We remove MaxSAT reasoning and/or IncUB from IncMaxCLQ to obtain three weakened versions of IncMaxCLQ. IncMaxCLQ is then compared with these weakened versions to analyze the impact of MaxSAT reasoning, IncUB and their combination. The performance of IncMaxCLQ is also compared with several state-of-the-art exact solvers for MaxClique. The experimentation has been performed on an Intel Xeon CPU E7-8837@2.67GHz under Linux with 4GB of memory and 24MB of cache memory (shared by 8 running cores). All solvers were compiled using gcc/g++ -O3. All runtimes are in seconds.

We first describe the benchmarks used and the solvers compared, then discuss and analyze the experimental results.

A. Benchmarks

Although MaxClique is an important NP-hard problem, the state-of-the-art of MaxClique solving is not as advanced as in SAT and CSP. There is no MaxClique solver competition as in SAT and CSP, apart from the DIMACS challenge held in 1992. State-of-the-art exact MaxClique solvers are proposed in the literature usually by showing their performance on random and/or DIMACS graphs[1] [21], [22], [23], [24], [10], [16], [4], [7], [20], [11], [12].

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

DIMACS graphs fall into three categories, similarly to SAT instances in SAT competitions: (1) Graphs generated using randomness and some properties, (2) Crafted graphs with special structures, (3) Graphs from real world problems.

Recently, it is recommended in [5], [8] to use the `frb⋆` graphs[2] to evaluate MaxClique and VC (Vertex Cover) solvers. The `frb⋆` graphs are encoded from CSP instances generated using the CSP model RB [26], [27] and are harder than most DIMACS graphs. They are currently rather used to evaluate heuristic MaxClique and VC solvers [8], [19], [5], [2], because they are very hard for exact solvers. As indicated in [9], [17], state-of-the-art exact solvers can only solve the few smallest and easiest `frb⋆` instances in reasonable time.

In this paper, we use random graphs, DIMACS graphs and frb* graphs to evaluate our solvers, which are essential benchmarks used in [17] to study MaxClique solvers. We also considered the 11 protein alignment instances used in [9], which are solved by all tested solvers within 2 seconds.

B. Solvers

In order to identify what makes our solver IncMaxCLQ efficient, we compare it with its three weakened versions:

- IncMaxCLQ$_{\backslash MaxSAT}$
- IncMaxCLQ$_{\backslash IncUB}$
- IncMaxCLQ$_{\backslash MaxSAT \backslash IncUB}$

which are obtained respectively by removing UB$_{MaxSAT}$, IncUB, or both in line 6 of Algorithm 1[3]. The comparison of these versions of IncMaxCLQ allows to identify the individual impact of UB$_{MaxSAT}$ and IncUB, and to show their complementarity, since the only difference of these versions is that they use or not UB$_{MaxSAT}$ and/or IncUB in line 6 of Algorithm 1 when computing vertexUB[v]. All these versions of IncMaxCLQ use the same vertex ordering computed by Algorithm 2.

In addition, we also compare IncMaxCLQ with the following state-of-the-art solvers:

- MaxCLQ[4]: We use the version presented in [12] and improved from [11]. MaxCLQ is faster than MCR, MaxCliqueDyn, and Regin's solver [20], according to [12], [11]. MaxCLQ is the same as IncMaxCLQ$_{\backslash IncUB}$ but uses a dynamic vertex ordering;
- MaxCliquedyn[5], version dec. 2012. It is improved from the 2010 version which was better than MCR [11];
- MCS [24], MCS is improved from MCR. The original codes of MCS are not open. We use a version implemented by Wang Kai, which has similar performance to the original version [25];

[2]BHOSLIB benchmark available at http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

[3]We use the term "$\backslash E$" to indicate that the element E has been removed from IncMaxCLQ

[4]available at http://home.mis.u-picardie.fr/~cli/EnglishPage.html

[5]available at http://www.sicmm.org/~konc/maxclique/

- Cliquer[6], version 1.21 released in early 2010.

C. *Experimental results for random and DIMACS graphs*

The advantage of random graphs is that they allow to show asymptotic behavior of a solver. Table II compares the median run times of IncMaxCLQ with its weakened versions: IncMaxCLQ$_{\backslash MaxSAT\backslash IncUB}$, IncMaxCLQ$_{\backslash MaxSAT}$, IncMaxCLQ$_{\backslash IncUB}$, for random graphs of different numbers of vertices (N) and densities (D). The cutoff time is 2000 seconds for each instance. At each point we generate 51 graphs and show the median runtime in seconds of each solver for these graphs (the 51 runtimes are sorted in the increasing ordering and the 26th runtime is the median). To save space, for a given N, we do not display neither the results for too easy densities (i.e., the median runtime at the densities is smaller than 20 seconds for all solvers tested in the paper), nor the results for too hard densities (i.e., no solver can solve 26 instances within the cutoff time).

For very sparse graphs (i.e. density$\leq$0.3), all the 4 versions of IncMaxCLQ have similar performance. However, for other graphs, Table II clearly shows that integrating UB$_{MaxSAT}$ and IncUB into the basic version IncMaxCLQ$_{\backslash MaxSAT\backslash IncUB}$ allows to improve its performance, and that UB$_{MaxSAT}$ and IncUB are complementary. Recall that the only difference of these versions of IncMaxCLQ is that they use or not UB$_{MaxSAT}$ and/or IncUB. In general the gain due to UB$_{MaxSAT}$ and IncUB increases with the size and the density of graphs. Algorithm 2 makes all versions of IncMaxCLQ use the degeneracy ordering for random graphs.

N	D	IncMC $\backslash MS\backslash IncUB$	IncMC $\backslash MS$	IncMC $\backslash IncUB$	IncMC
150	0.90	18.19	8.37	0.33	**0.17**
150	0.95	14.50	4.66	0.04	**0.01**
200	0.80	21.18	11.53	2.39	**1.60**
200	0.90	1390	524.8	15.46	**7.60**
200	0.95	-	1228	3.29	**1.26**
250	0.80	266.9	132.6	25.55	**16.79**
250	0.90	-	-	555.1	**237.2**
250	0.95	-	-	389.8	**123.6**
300	0.80	1869	944.0	206.7	**120.3**
500	0.60	85.23	54.45	45.12	**33.12**
500	0.70	-	-	1623	**1348**
600	0.60	448.6	345.2	226.6	**201.1**
1000	0.40	19.23	13.43	16.34	**11.12**
1000	0.50	520.2	351.2	376.6	**270.3**
2000	0.30	47.23	41.65	46.49	**39.95**
2000	0.40	1652	1377	1372	**1174**
5000	0.10	8.30	**8.23**	8.43	8.38
5000	0.20	151.2	**130.5**	152.4	139.4
10000	0.05	25.65	25.43	25.65	**25.19**
10000	0.10	67.12	66.50	66.23	**66.19**

Table II
MEDIAN RUNTIMES IN SECONDS FOR RANDOM GRAPHS. INCMC STANDS FOR INCMAXCLQ, MS FOR MAXSAT, "-" MEANS THAT THE SOLVER SOLVES FEWER THAN 26 INSTANCES SO THAT THE MEDIAN CANNOT BE COMPUTED.

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

Table III compares IncMaxCLQ with Cliquer, MaxCliqueDyn, MCS and MaxCLQ on random graphs in the same condition as in Table II. IncMaxCLQ is substantially better than MaxCLQ especially for sparse graphs. Note that MaxCLQ and IncMaxCLQ use the same data structure and the same UB$_{MaxCLQ}$. Their difference lies in the use of IncUB and the static vertex ordering enabling IncUB in IncMaxCLQ. IncMaxCLQ has comparable performance to MCS and MaxCliqueDyn for sparse graphs and is substantially better for dense graphs.

N	D	Cliquer	Dyn	MaxCLQ	MCS	IncMC
150	0.90	320.3	1.27	0.25	0.79	**0.17**
150	0.95	560.6	0.35	0.02	0.16	**0.01**
200	0.80	99.57	3.71	1.81	2.59	**1.60**
200	0.90	-	65.16	10.23	32.31	**7.60**
200	0.95	-	47.91	2.05	22.78	**1.26**
250	0.80	-	36.90	19.19	31.07	**16.79**
250	0.90	-	-	370.3	1898	**237.2**
250	0.95	-	-	194.2	-	**123.6**
300	0.80	-	342.0	158.5	289.2	**120.3**
500	0.60	167.6	28.79	38.34	**27.65**	33.12
500	0.70	-	1180	**1054**	1188	1348
600	0.60	1164	151.7	197.8	**142.2**	201.1
1000	0.40	19.27	9.84	25.78	**9.40**	11.12
1000	0.50	774.3	212.2	413.9	**209.1**	270.3
2000	0.30	51.28	28.62	145.5	**26.43**	39.95
2000	0.40	-	**766.3**	-	767.5	1174
5000	0.10	**1.81**	3.49	67.91	2.56	8.38
5000	0.20	116.4	93.71	-	**86.36**	139.4
10000	0.05	**3.62**	25.88	298.5	5.47	25.19
10000	0.10	36.62	97.16	-	**31.97**	66.19

Table III
MEDIAN RUNTIMES OF INCMAXCLQ AND FOUR STATE-OF-THE-ART SOLVERS FOR RANDOM GRAPHS. DYN STANDS FOR MAXCLIQUEDYN.

Table IV compares IncMaxCLQ with its three weakened versions (columns 5–8) and with the state-of-the-art solvers (columns 8–12) on DIMACS graphs solved by at least one solver within 2000 seconds (excluding the instances that all solvers solve in less than 10 seconds). UB$_{MaxSAT}$ and IncUB are clearly complementary for the performance of IncMaxCLQ, making IncMaxCLQ the most robust solver for DIMACS graphs solving all instances in Table IV. Algorithm 2 makes all versions of IncMaxCLQ use the degeneracy ordering for branching, except for the highly structured graphs `keller.5` and hamming10-2, for which the MaxIndSet vertex ordering is used for branching.

Table V compares IncMaxCLQ with its weakened versions on the *frb** graphs, showing once again the complementarity of UB$_{MaxSAT}$ and IncUB in IncMaxCLQ. Algorithm 2 makes all versions of IncMaxCLQ use the MaxIndSet ordering for branching. IncMaxCLQ is substantially better than Cliquer MaxCliqueDyn, MaxCLQ and MCS, as shown in Table VI.

VII. **Conclusion**

We have presented the combining of MaxSAT reasoning and an incremental upper bound in branch-and-bound

Graph	N	D	ω	IncMC $\backslash MS \backslash IncUB$	IncMC $\backslash MS$	IncMC $\backslash IncUB$	IncMC	Cliquer	Dyn	MaxCLQ	MCS
brock400_1	400	74	27	-	1230	581.2	**277.9**	-	527.1	417.4	469.9
brock400_2	400	74	29	-	913.8	530.9	210.9	-	226.0	**130.9**	205.8
brock400_3	400	74	31	-	1100	436.8	259.9	1572	429.0	**126.5**	316.3
brock400_4	400	74	33	1853	882.7	344.5	197.7	233.9	210.7	**154.7**	170.9
C250.9	250	89	44	-	-	521.1	**333.2**	-	-	363.1	-
DSJC1000.5	1000	50	15	524.8	337.8	381.2	261.2	763.1	209.7	382.8	**209.4**
gen200_p0.9_55	200	90	55	7.95	1.53	0.88	0.34	64.34	0.71	**0.06**	0.80
gen400_p0.9_55	400	90	55	1050	108.1	3.13	**1.75**	-	-	-	-
gen400_p0.9_75	400	90	75	1911	324.1	0.81	**0.65**	-	-	1787	-
hamming10-2	1024	99	512	56.92	25.81	59.42	33.17	0.16	54.55	**0.06**	0.25
keller5	776	75	27	-	-	741.3	**177.4**	-	-	-	-
MANN_a27	378	99	126	66.66	7.10	2.57	0.43	-	-	**0.16**	0.43
MANN_a45	1035	99	345	-	-	-	114.1	-	-	**24.78**	78.99
p_hat1000-2	1000	49	46	961.9	633.6	85.84	**79.19**	-	314.6	296.4	159.7
p_hat1500-1	1500	25	12	7.39	6.32	6.32	5.14	7.58	3.24	12.26	**2.96**
p_hat300-3	300	74	36	6.33	4.18	0.94	**0.87**	451.2	3.38	1.40	1.76
p_hat500-2	500	50	36	1.93	0.63	0.40	**0.32**	122.3	1.14	0.85	0.52
p_hat500-3	500	75	50	875.4	505.5	43.93	**29.61**	-	257.1	105.2	94.57
p_hat700-2	700	49	44	15.47	11.99	1.41	**1.25**	-	9.62	6.15	3.97
p_hat700-3	700	74	62	-	-	595.8	**357.4**	-	-	1383	1918
san200_0.9_2	200	90	60	0.63	0.27	0.68	0.26	10.44	0.62	0.16	**0.03**
san200_0.9_3	200	90	44	24.57	6.41	0.65	0.27	437.6	2.47	0.89	**0.04**
san400_0.7_3	400	70	22	21.90	11.41	5.66	3.91	3.99	1.69	3.79	**0.94**
san400_0.9_1	400	90	100	1404	15.64	1.92	**0.29**	-	18.43	32.66	-
sanr200_0.9	200	89	42	480.5	193.4	6.27	**3.34**	-	36.48	7.61	25.04
sanr400_0.7	400	70	21	522.3	378.2	152.5	136.7	-	125.2	**121.6**	122.6

Table IV

RUNTIMES IN SECONDS FOR DIMACS INSTANCES THAT ARE SOLVED BY AT LEAST ONE SOLVER WITHIN 2000 SECONDS, EXCLUDING THOSE SOLVED BY ALL SOLVERS IN LESS THAN 10 SECONDS. MS STANDS FOR MAXSAT, INCMC FOR INCMAXCLQ, AND DYN FOR MAXCLIQUEDYN. "-" MEANS THAT THE SOLVER CANNOT SOLVE THE INSTANCE WITHIN 2000 SECONDS.

algorithms for MaxClique. The incremental upper bound IncUB is derived in linear time from upper bounds for subgraphs organized in static degeneracy or MaxIndSet vertex ordering. We implemented the approach in a highly efficient MaxClique solver called IncMaxCLQ and compared its performance with its three weakened versions to show the complementarity of IncUB and MaxSAT reasoning. Indeed, a deeper analysis of the results shows that the search tree size of IncMaxCLQ is significantly smaller than that of IncMaxCLQ$_{\backslash IncUB}$ (not shown here due to the lack of space), meaning that IncUB is often tighter than UB$_{MaxSAT}$, other things being equal in these versions of IncMaxCLQ. IncUB can also be combined with other upper bounds such as UB$_{IndSet}$ to improve them, as showed by the superiority of IncMaxCLQ$_{\backslash MaxSAT}$ over IncMaxCLQ$_{\backslash MaxSAT \backslash IncUB}$, in which UB$_{IndSet}$ is used without MaxSAT reasoning, and the only difference is that they use or not IncUB.

Acknowledgment

We would like to thank Felip Manya, Thomas Schiex and anonymous reviewers for their helpful comments and suggestions. This work was supported by Chinese State Key Laboratory of Software Development Environment Open Fund under grant No. SKLSDE-2012KF-07.

REFERENCES

[1] Barnes, E., Strickland, D.M., Sokol, J.S.: Optimal protein structure alignment using maximum cliques. Operations Research 53, 389–402 (2005)

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

[3] P. Berman and A. Pelc, Distributed Fault Diagnosis for Multiprocessor Systems, Proceedings of the 20th Annual International Symposium on Fault-Tolerant Computing, Newcastle, UK, 340–346 (1990)

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

[5] Shaowei Cai, Kaile Su, Abdul Sattar.: Local search with edge weighting and configuration checking heuristics for minimum vertex cover. In:*Artificial Intelligence*, Vol. 175, pp. 1672–1696, 2011.

[6] Eppstein D.; Darren, S. Listing all maximal cliques in large sparse real-world graphs. In Experimental Algorithms, LNCS 6630. Comput. Sci. 2011, 6630, 364–375.

[7] T. Fahle.: Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In: *Proc. of ESA-2002*, pp. 485–498, 2002.

[8] A. Grosso, M. Locatelli, W. Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem, J Heuristics (2008) 14: 587–612

[9] F. Heras, J. Larrosa: A Max-SAT Inference-Based Preprocessing for Max-Clique. In proceeedings of SAT'2008, LNCS 4996, pp139–152

Graph frb*	N	D	IncMC $\backslash MS \backslash IncUB$	IncMC $\backslash MS$	IncMC $\backslash IncUB$	IncMC
frb30-15-1	450	82	0.94	0.62	0.28	**0.16**
frb30-15-2	450	82	1.51	0.73	0.28	**0.18**
frb30-15-3	450	82	4.48	0.38	0.62	**0.18**
frb30-15-4	450	82	3.94	0.19	0.95	**0.17**
frb30-15-5	450	82	1.15	0.93	0.87	**0.18**
frb35-17-1	595	84	42.52	6.10	1.28	**1.27**
frb35-17-2	595	84	127.2	25.20	2.11	**1.11**
frb35-17-3	595	84	13.50	2.45	0.93	**0.55**
frb35-17-4	595	84	7.90	1.88	0.75	**0.42**
frb35-17-5	595	84	30.27	6.54	1.26	**0.72**
frb40-19-1	760	85	183.8	26.75	3.90	**2.20**
frb40-19-2	760	85	12.12	2.26	1.39	**0.82**
frb40-19-3	760	85	215.9	42.11	3.83	**1.74**
frb40-19-4	760	85	1201	231.9	11.98	**7.58**
frb40-19-5	760	85	596.4	98.76	9.84	**6.99**
frb45-21-1	945	86	-	2756	228.6	**131.5**
frb45-21-2	945	86	-	1166	130.6	**77.56**
frb45-21-3	945	86	-	934.8	83.59	**46.49**
frb45-21-4	945	86	-	916.5	63.96	**38.42**
frb45-21-5	945	86	-	-	364.1	**236.8**
frb50-23-1	1150	87	-	-	1528	**870.3**
frb50-23-2	1150	87	-	-	554.9	**380.4**
frb50-23-4	1150	87	2480	325.4	34.79	**18.80**
frb50-23-5	1150	87	-	-	495.8	**270.6**
frb53-24-2	1272	88	-	-	371.4	**220.7**
frb53-24-3	1272	88	-	-	-	**2321**
frb53-24-5	1272	88	-	-	1953	**1113**
frb56-25-2	1400	88	-	-	-	**2952**

Table V
RUNTIMES IN SECONDS OF INCMAXCLQ AND ITS WEAKENED VERSIONS FOR *frb** GRAPHS SOLVED BY AT LEAST ONE SOLVER. THE CUTOFF TIME IS 3600 SECONDS. MS STANDS FOR MAXSAT, INCMC FOR INCMAXCLQ. "-" MEANS THAT THE SOLVER CANNOT SOLVE THE INSTANCE WITHIN 3600 SECONDS. THE CARDINALITY OF A MAXIMUM CLIQUE IN A GRAPH FRBω-δ-α IS ω.

Graph	Cliquer	Dyn	MaxCLQ	MCS	IncMC
frb30-15-1	23.99	3126	643.7	1241	**0.16**
frb30-15-2	49.45	-	1042	1790	**0.18**
frb30-15-3	187.9	3427	484.3	958.1	**0.18**
frb30-15-4	1.66	-	1306	3276	**0.17**
frb30-15-5	0.23	-	837.7	1387	**0.18**
frb35-17-1	22.76	-	-	-	**1.27**
frb35-17-2	94.09	-	-	-	**1.11**
frb35-17-3	-	-	-	-	**0.55**
frb35-17-4	-	-	-	-	**0.42**
frb35-17-5	173.9	-	-	-	**0.72**
frb40-19-1	-	-	-	-	**2.20**
frb40-19-2	-	-	-	-	**0.82**
frb40-19-3	410.2	-	-	-	**1.74**
frb40-19-4	2743	-	-	-	**7.58**
frb40-19-5	73.93	-	-	-	**6.99**
frb45-21-1	-	-	-	-	**131.5**
frb45-21-2	-	-	-	-	**77.56**
frb45-21-3	-	-	-	-	**46.49**
frb45-21-4	-	-	-	-	**38.42**
frb45-21-5	-	-	-	-	**236.8**
frb50-23-1	-	-	-	-	**870.3**
frb50-23-2	-	-	-	-	**380.4**
frb50-23-4	3269	-	-	-	**18.80**
frb50-23-5	-	-	-	-	**270.6**
frb53-24-2	-	-	-	-	**220.7**
frb53-24-3	-	-	-	-	**2321**
frb53-24-5	-	-	-	-	**1113**
frb56-25-2	-	-	-	-	**2952**

Table VI
RUNTIMES IN SECONDS OF INCMAXCLQ AND STATE-OF-THE-ART SOLVERS FOR *frb** GRAPHS SOLVED BY AT LEAST ONE SOLVER. INCMC STANDS FOR INCMAXCLQ, DYN FOR MAXCLIQUEDYN. "-" MEANS THAT THE SOLVER CANNOT SOLVE THE INSTANCE WITHIN 3600S.

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

[11] Chu Min Li and Zhe Quan.: An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In: *Proc. of the 24th AAAI*, pages 128–133, 2010.

[12] Chu Min Li and Zhe Quan. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: *Proc. of the 22th ICTAI*, pages 344–351, 2010.

[13] Li, C.M, Zhu, Z., Manya, F, Simon, L. Optimizing with Minimum Satisfiability, In Artificial Intelligence 190 (2012) 32–44.

[14] Li, C.M, Zhu, Z., Manya, F, Simon, L. Minimum satisfiability and its applications, In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI11), pp 605–610, 2011

[15] D. R. Lick and A. T. White. k-degenerate graphs. Canad. J. Math. 22:1082–1096, 1970, www.smc.math.ca/cjm/v22/p1082.

[16] P. R. J. Ostergard. A fast algorithm for the maximum clique problem. In: *Discrete Applied Mathematics* 120 (2002), 197–207.

[17] P. Prosser, Exact Algorithms for Maximum Clique: A Computational Study, Algorithms 2012, 5, 545–587

[18] W. Pullan, H. H. Hoos. Dynamic Local Search for the Maximum Clique Problem. In: *Journal of Artificial Intelligence Research*, Vol. 25, pp. 159–185, 2006.

[19] W. Pullan, F. Mascia, M. Brunato. Cooperating local search for the maximum clique problem. J Heuristics (2011) 17: 181–199

[20] J.C. Regin, Using Constraint Programming to Solve the Maximum Clique Problem. In Proceedings of CP2003, LNCS 2833, Kinsale, Ireland, 2003; pp. 634–648.

[21] P. S. Segundo, R. L. Diego, J. Augustin. An exact bit-parallel algorithm for the maximum clique problem. Comput. Oper. Res 2011, 38, 571–581.

[22] Tomita, E.; Sutani, Y.; Higashi, T.; Takahashi, S.; Wakatsuki, M. An Efficient Branch-and-Bound Algorithm for Finding a Maximum Clique. In Proc. of DMTCS 2003, LNCS 2731, France, 2003; pp. 278–289.

[23] E. Tomita, T. Kameda.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. In: *J. Glob Optim*, 37:95–111, 2007.

[24] Tomita, E.; Sutani, Y.; Higashi, T.; Takahashi, S.; Wakatsuki, M. A Simple and Faster Branch-and-Bound Algorithm for Finding Maximum Clique. In Proc. of WALCOM 2010, LNCS 5942, Bangladesh, 2010; pp. 191–203.

[25] K. Wang, Personal Communication.

[26] K. Xu, Frédéric Boussemart, Fred Hemery and Christophe Lecoutre.: Random Constraint Satisfaction: Easy Generation of Hard (Satisfiable) Instances. In: *Artificial Intelligence*, 171:514–534, 2007.

[27] K. Xu and W. Li.: Exact Phase Transitions in Random Constraint Satisfaction Problems. In: *Journal of Artificial Intelligence Research*, 12:93–103, 2000.