2010 22nd International Conference on Tools with Artificial Intelligence

Combining Graph Structure Exploitation and Propositional Reasoning for the Maximum Clique Problem

Chu-Min LI
Université de Picardie Jules Verne, Amiens, France
http://www.laria.u-picardie.fr/~cli/
Email: chu-min.li@u-picardie.fr

Zhe Quan
Université de Picardie Jules Verne, Amiens, France
Email: quanzhe@gmail.com

***Abstract*—State-of-the-art branch-and-bound algorithms for the maximum clique problem (Maxclique) generally exploit the structural information of a graph G to partition G into independent sets, in order to derive an upper bound for the cardinality of a maximum clique of G, which cannot be very tight for imperfect graphs. On the other hand, while Maxclique can be easily encoded into MaxSAT to be solved using a MaxSAT solver, general-purpose MaxSAT solvers are not competitive for solving Maxclique, because they do not exploit the structural information of the graph. Recently, we have shown that propositional reasoning developed for the MaxSAT solvers can be used to improve the upper bound based on the partition. In this paper, we propose and study several improvements to this approach by better combining graph structure exploitation and propositional reasoning to solve Maxclique. Experimental results show that the improvements are very effective on hard random graphs and on DIMACS Maxclique benchmarks which are widely used to evaluate branch-and-bound algorithms for Maxclique.**

***Keywords*-Branch-and-Bound, Maxclique, MaxSAT**

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. Edge (v_i, v_j) with $i \neq j$ is said to connect vertices v_i and v_j. A clique of G is a subset C of V such that every two vertices in C are connected by an edge. The maximum clique problem (Maxclique for short) consists in finding a clique of G of the largest cardinality (denoted $\omega(G)$).

An independent set of G is a subset S of V such that no two vertices in S are connected. A clique of G is an independent set of its complement graph $\overline{G}$, and vice versa, where $\overline{G}$ is defined to be (V, $\overline{E}$) with $\overline{E}$=$\{(v_i, v_j) \mid (v_i, v_j) \in V \times V, i \neq j$ and $(v_i, v_j) \notin E)\}$. Given G, the Graph Coloring Problem (GCP) asks to find the minimum number of colors (i.e., the chromatic number $\chi(G)$) necessary to color the vertices of G such that no two connected vertices share the same color. Solving GCP is equivalent to partitioning G into a minimum number of independent sets, because all vertices sharing the same color in G constitute an independent set. We have $\chi(G) \geq \omega(G)$, since each vertex in a clique should be assigned a different color. A graph G is perfect if $\chi(G')=\omega(G')$ for any induced subgraph G' of G. The GCP, Maxclique, and maximum independent set problem can all be solved in polynomial time for a perfect graph [6].

Maxclique is a very important combinatorial problem, because it appears in many real-world applications including project selection, classification theory, fault tolerance, coding theory, computer vision, economics, information retrieval, signal transmission theory, aligning DNA and protein sequences, and other specific problems. Although Maxclique can be solved in polynomial time for a perfect graph [6], it is NP-hard in general. A huge amount of effort has been devoted to solve it. See [17] which gives a survey of the early intensive work on Maxclique.

In the literature, we mainly distinguish two types of algorithms for Maxclique: approximation methods including stochastic local search algorithms, e.g. [18], and exact algorithms including branch-and-bound algorithms, e.g. [14], [20], [8], [16], [2], [4], [19]. Approximation algorithms are able to solve large and hard Maxclique problems but cannot guarantee the optimality of their solutions. Exact algorithms guarantee the optimality of the solutions they find. In this paper, we focus on branch-and-bound algorithms for Maxclique.

State-of-the-art branch-and-bound algorithms for Maxclique generally exploit the structural information of a graph G to partition G into independent sets, in order to derive an upper bound for the cardinality of a maximum clique of G. There are two drawbacks in this approach: (i) the number of independent sets in a partition of G generally is not minimum, since the partition problem itself is NP-hard; (ii) even if the partition is minimum, there can be a large difference between $\chi(G)$ and $\omega(G)$ when G is not perfect, so that $\chi(G)$ is not a tight upper bound of $\omega(G)$. The two drawbacks might probably explain the stagnation of the research on exact algorithms for Maxclique.

Given a set of Boolean variables $\{x_1,x_2,...,x_n\}$, a literal l is a variable x_i or its negation $\bar{x}_i$, a clause is a logical *or* of literals. A CNF formula ϕ is a set of clauses. The MaxSAT problem asks to find an assignment of truth values (0 or 1) to the Boolean variables to maximize the number of satisfied clauses in ϕ. Maxclique can be encoded into MaxSAT and then solved using a MaxSAT solver. Unfortunately, MaxSAT solvers are not competitive to directly solve Maxclique, because they do not exploit the structural properties of the graph.

1082-3409/10 $26.00 © 2010 IEEE
DOI 10.1109/ICTAI.2010.57

Recently, we have shown that propositional reasoning developed for MaxSAT solvers can be used to improve the upper bound based on the partition of a graph G [14]. We first defined a new encoding from Maxclique into MaxSAT based on the partition and then used a simple MaxSAT technology to improve the upper bound. The resulted branch-and-bound algorithm for Maxclique, called MaxCLQ, is substantially faster than previous branch-and-bound algorithms without MaxSAT technology and MaxSAT solvers for Maxclique. To the best of our knowledge, this was the first time that the strength of specific algorithms for Maxclique in exploiting the structural information of a graph and the strength of MaxSAT technology in propositional reasoning are combined to solve Maxclique.

However, the approach presented in [14], although very powerful, is rather preliminary. On the one hand, although the best branch-and-bound algorithms for MaxClique in our knowledge, MCQ [21], MCR [20], MaxCliqueDyn [8], and MaxCLQ, all use the same coloring process introduced in MCQ to partition a graph into independent sets in their upper bound computation, it appears that MaxCliqueDyn exploits in a better way the structural information of the graph in the graph partitioning, making MaxCliqueDyn generally faster than MaxCLQ$^-$, the version of MaxCLQ without propositional reasoning [14]. So, the graph structure exploitation of MaxCLQ should be improved. On the other hand, the propositional reasoning presented in [14] to improve the upper bound obtained using the coloring process is very simple and limited, which can obviously be improved. We present several improvements to MaxCLQ in this paper.

This paper is organized as follows. In Section II, we present some preliminaries and a basic branch-and-bound algorithm for Maxclique. In Section III, we review the upper bounds for the cardinality of a maximum clique of a graph G based on a partition of G into independent sets, the encodings of Maxclique into MaxSAT that make propositional reasoning applicable to solve Maxclique, and the propositional reasoning presented in [14] to improve the upper bounds given by the partition. In Section IV, we present three improvements to the approach presented in [14], each being evaluated in Section V using random graphs and the DIMACS Maxclique benchmark[1]. The experimental results show that the improvements are very effective.

II. Searching for a Maximum Clique in a Graph

Consider a graph G=(V, E). Let V' be a subset of V, the subgraph G' of G induced by V' is defined as G'=(V', E'), where E'=$\{(v_i, v_j) \in E \mid v_i, v_j \in V'\}$. The set of 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_v denotes the subgraph induced by $\Gamma(v)$, and $G\backslash v$ denotes the subgraph induced by $V\backslash\{v\}$. $G\backslash v$ is obtained by removing v and all edges connecting v from G. The density of a graph of n vertices and m edges is computed as $2m/(n(n-1))$. G can have several maximum cliques.

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

Algorithm 1: MaxCLQ(G, C, LB), a branch and bound algorithm for Maxclique

Input: A graph G=(V, E), a clique C, and the cardinality LB of the largest clique found so far
Output: $C \cup C'$, where C' is a maximum clique of G, if $|C \cup C'| > LB$, $\emptyset$ otherwise

```
1  begin
2      if |V|=0 then return C;
3      UB ← overestimation(G)+|C|;
4      if LB ≥ UB then return ∅;
5      select a vertex v from G of the minimum degree;
6      C1 ← MaxCLQ(Gv, C∪{v}, LB);
7      if |C1| > LB then LB ← |C1| ;
8      C2 ← MaxCLQ(G\v, C, LB);
9      if |C1| ≥ |C2| then return C1; else return C2;
10 end
```

A graph is perfect if it and its complement graph do not contain an induced cycle of odd length at least 5 [3]. Therefore, the smallest graph which is not perfect is a cycle of length five as displayed in Figure 1. It is clear that $\chi(G)$ is three but a maximum clique only contains two vertices.

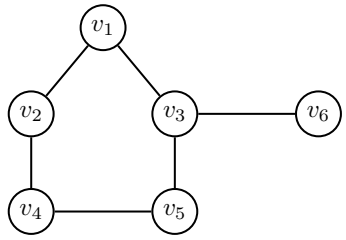

Fig. 1. A simple imperfect graph ($\chi(G)$=3 and $\omega(G)$=2)

In [14], we have presented a branch-and-bound algorithm called MaxCLQ to search for a maximum clique in a graph G. Its basic form is inspired by the branch-and-bound algorithm MaxSatz for MaxSAT [11] and is shown in Algorithm 1. In [14], the principle of this algorithm is illustrated using the graph G in Figure 1. Initially C=$\emptyset$ and LB=0, and MaxCLQ(G, $\emptyset$, 0) returns a maximum clique ($\{v_3, v_6\}$) of G as follows.

In line 3, the function overestimation(G) gives an upper bound for the cardinality of a maximum clique in G which is clearly larger than 0, since G is not empty. Then MaxCLQ chooses v_6 as the branching vertex (line 5), since it is of the minimum degree, so that all cliques of G are implicitly divided into two sets: the set of cliques containing v_6 and the set of cliques not containing v_6. The first recursive call (line 6) MaxCLQ(G_{v_6}, $\{v_6\}$, 0) returns a clique ($\{v_3, v_6\}$) containing v_6, where G_{v_6}=($\{v_3\}$, $\emptyset$).

Then in line 7, the lower bound LB becomes 2. The second recursive call MaxCLQ($G\backslash v_6$, $\emptyset$, 2) tries to find a clique not containing v_6 and larger than 2 in $G\backslash v_6$ (line 8), where $G\backslash v_6$ is the cycle consisting of $\{v_1,v_2,v_3,v_4,v_5\}$. If overestimation($G\backslash v_6$) uses a coloring process and returns

an upper bound 3 (recall that $\chi(G\backslash v_6)$=3) for a maximum clique in $G\backslash v_6$, MaxCLQ($G\backslash v_6$, $\emptyset$, 2) has to make two further recursive calls by choosing a branching vertex, e.g. v_1: MaxCLQ($(G\backslash v_6)_{v_1}$, $\{v_1\}$, 2) and MaxCLQ($(G\backslash v_6)\backslash v_1$, $\emptyset$, 2). The execution of both calls returns $\emptyset$ (do it to see this). So, MaxCLQ($G\backslash v_6$, $\emptyset$, 2) returns $\emptyset$ in line 9. However, if overestimation($G\backslash v_6$) can return a tighter upper bound, so that UB=LB and MaxCLQ($G\backslash v_6$, $\emptyset$, 2) directly returns $\emptyset$ (line 4) without further recursive calls, since UB=LB means that a clique of larger cardinality cannot be found.

Finally, MaxCLQ(G, $\emptyset$, 0) returns C_1=$\{v_3, v_6\}$ in line 9.

Generally speaking, MaxCLQ develops a binary search tree. When it branches on a vertex v, it implicitly divides all cliques of a graph into two sets: those containing v and those not containing v, and searches for the largest clique in each set for which the UB is bigger than the current LB. To the best of our knowledge, this is the first presentation of a branch-and-bound algorithm for MaxClique as a binary search tree. We believe that this presentation is easier to understand than the n-ary search tree presentation of branch-and-bound algorithms for Maxclique in [20], [8], [16], [2], [4], [19].

Note that when MaxCLQ selects a vertex v for branching, the number of vertices in the subgraph $G\backslash v$ is the same for any vertex v. However, G_v is the smallest if v is of the minimum degree, presumably allowing a quick computation of the largest clique in G_v, which is the intuition behind the selection of the vertex of the minimum degree for branching in MaxCLQ.

III. Exploitation of Graph Structure and Propositional Reasoning in Branch-and-Bound Algorithms for Maxclique

On the one hand, most branch-and-bound algorithms for Maxclique compute the upper bound for the cardinality of a maximum clique of a graph by exploiting the structural information of the graph. On the other hand, Maxclique can be easily encoded as partial MaxSAT, so that modern MaxSAT solvers can solve a Maxclique instance using propositional reasoning. This section reviews graph structure exploitation, propositional reasoning, and their combination for Maxclique.

A. Upper bounds for Maxclique based on a partition of a graph into independent sets

The best graph structure exploitation in our knowledge for upper bound computation is the partition of the graph into independent sets introduced in MCQ [21], and also used in MCR [20], Maxcliquedyn [8], and MaxCLQ, shown in Algorithm 2.

The upper bound obtained using partition(G) is $|P|$, i.e., the number of independent sets in the returned partition. The difference between MCQ, MCR and Maxcliquedyn is in the order in which the vertices are inserted into an independent set.

MCQ inserts vertices into an independent set in a predetermined order. Suppose that the current independent sets are S_1, S_2, ..., S_k (in this order, k is 0 at the beginning of the coloring process), MCQ inserts the current first vertex v into the first S_i such that v is non-connected to all vertices already in S_i. If such a S_i does not exist, a new independent set S_{k+1} is opened and v is inserted here. After all vertices are partitioned into independent sets, they are reordered according to their independent set, vertices in S_i coming before vertices in S_j if $i < j$. This coloring process is executed for G_v after each branching on the vertex v. The predetermined order of vertices in G_v is inherited from G. MCR improves MCQ with a better initial order of vertices in the initial input graph.

Algorithm 2: partition(G), a partition of G into independent sets

```
Input: A graph G=(V, E)
Output: A partition of G into independent sets
1  begin
2      P ← ∅;
3      while G is not empty do
4          v ← the vertex of G selected using an order;
5          remove v from G;
6          if there is an independent set S in P in which v
           is not connected to any vertex then
7              insert v into S;
8          else
9              create a new independent set S = {v};
10             P ← P ∪ {S};
11     return P;
12 end
```

MaxCliqueDyn is also improved from MCQ. While MCQ (as well as MCR) only computes the degree of vertices at the root of the search tree for the initial input graph, MaxCliqueDyn dynamically recomputes the degree of vertices at the nodes near the root of the search tree chosen using a parameter, and re-orders the vertices in the decreasing order of their degree before partitioning these vertices. Concretely, Maxcliquedyn maintains two global variables $level$ and $S[level]$, where $level$ corresponds to the number of recursive calls MaxCLQ(G_v, $C\cup\{v\}$, LB) (line 6 in Algorithm 1) in the path from the root to the current node, and $S[level]$ holds the sum of all steps the algorithm performs from the root up to and including the current $level$, each call MaxCLQ(G_v, $C\cup\{v\}$, LB) corresponding to a step, the second recursive call MaxCLQ($G\backslash v$, C, LB) (line 8 in Algorithm 1) to find a clique not containing v being not counted, neither in $level$, nor in $S[level]$.

At a search step of the level a, let $\#all\_steps$ denote the total number of steps of all levels from the beginning until the current step. Recall that Maxcliquedyn makes a depth-first search. A huge number of steps already done may be of a level higher than a. If $S[a]/\#all\_steps < T_{limit}$, where T_{limit} is a parameter, then the degree of all vertices is re-computed. Otherwise, the re-computation is not performed and the order of vertices is inherited from the parent step as in MCQ. So, if T_{limit}=0.025, the empirically best parameter value in

Maxcliquedyn, the re-computation of the vertex degree should be performed in roughly 2.5% of the search steps of the lowest levels (near the root).

The intuition behind the dynamic degree recomputation near the root of the search tree is that with precise degree information, the partition of a graph into independent sets presumably is of better quality, since vertices more constrained (i.e. with more neighbors) are inserted first into independent sets. However the degree computation is time-consuming. The parameter T_{limit}=0.025 gives a good trade-off for a broad range of graphs in Maxcliquedyn [8], making the algorithm to recompute the vertex degree in the most important 2.5% of the search steps.

MaxCLQ also uses Algorithm 2 to partition a graph into independent sets at every search tree node, except that the vertex degree is systematically re-computed at every node. Although the quality of the partition is presumably better (i.e., the partition presumably contains fewer independent sets), MaxCLQ^{-}, the version of MaxCLQ which does not use propositional reasoning to improve the upper bound, is slower than Maxcliquedyn, meaning that the graph structure exploitation in Maxcliquedyn is better than in MaxCLQ.

B. Encodings from Maxclique into partial MaxSAT

Given a MaxSAT problem ϕ, an assignment of truth values to Boolean variables satisfies a positive literal x if x=1, satisfies a negative literal $\bar{x}$ if x=0, satisfies a clause if at least one literal in the clause is satisfied. We specially distinguish unit clauses which contain only one literal, and empty clauses which do not contain any literal and cannot be satisfied. The MaxSAT problem ϕ is said to be partial, if some clauses of ϕ are hard, i.e., they should be satisfied in every solution, and the rest of clauses are soft and can be unsatisfied in a solution. A partial MaxSAT problem asks to find an assignment satisfying all the hard clauses and maximizing the number of satisfied soft clauses.

The usual way to encode a Maxclique instance G into a partial MaxSAT instance is to introduce a Boolean variable x for each vertex v of G, with the meaning that x=1 if and only if v is in the maximum clique. Then a hard clause $\bar{x}_i \vee \bar{x}_j$ is added for every pair of vertices v_i and v_j which are not connected, meaning that v_i and v_j cannot be in the same clique. It is clear that every assignment satisfying all the hard clauses gives a clique. Finally, a soft unit clause x is added for every vertex v. Maximizing the number of satisfied soft clauses while satisfying all the hard clauses gives a maximum clique.

For example, the Maxclique instance in Figure 1 can be encoded into a partial MaxSAT instance consisting of the soft clauses: $\{x_1, x_2, x_3, x_4, x_5, x_6\}$, and the hard clauses: $\{\bar{x}_1 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_5, \bar{x}_2 \vee \bar{x}_3, \bar{x}_2 \vee \bar{x}_5, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_6, \bar{x}_2 \vee \bar{x}_6, \bar{x}_4 \vee \bar{x}_6, \bar{x}_5 \vee \bar{x}_6\}$.

We proposed a new encoding from Maxclique into partial MaxSAT in [14] based on a partition of the graph into independent sets. In this new encoding, the Boolean variables and hard clauses are defined as in the usual encoding, but a soft clause is added for each independent set which is a logical *or* of the variables representing the vertices in the independent set. So, the usual encoding is just a particular case in which each independent set contains only one vertex. The new independent set based encoding contains substantially fewer soft clauses and is much more efficient than the usual encoding.

For example, the graph in Figure 1 can be partitioned into three independent sets $\{v_1, v_4, v_6\}$, $\{v_2, v_3\}$, $\{v_5\}$. Therefore, the independent set based MaxSAT encoding consists of the soft clauses: $\{x_1 \vee x_4 \vee x_6, x_2 \vee x_3, x_5\}$, and the same set of hard clauses as in the usual encoding.

Let ϕ be an independent set based MaxSAT encoding for a Maxclique instance G. The hard clauses oblige that at most one literal can be satisfied in a soft clause. Given an assignment satisfying all the hard clauses, a satisfied soft clause has exactly one satisfied literal because of the hard clauses. The corresponding independent set has one vertex in the clique given by the assignment. So, the set of variables evaluated to true in any optimal assignment give a maximum clique of G.

Unfortunately, modern MaxSAT solvers cannot compete the best specific algorithms for Maxclique such as MCR, Maxcliquedyn, and Regin, even using the more efficient independent set based encoding, at least for random graphs and DIMACS benchmarks, according to the experimental results reported in [9], [14]. Nevertheless, the independent set based encoding is more efficient than the usual encoding, which can be explained by the fact that the graph structure is taken into account in the encoding. The problem is that, when solving a partial MaxSAT instance resulted from any of the two encodings, the propositional reasoning in these MaxSAT solvers is not specially guided to exploit the graph structural information during the search, explaining the bad performance of MaxSAT solvers for Maxclique.

On the other hand, propositional reasoning is a low-level and fine-grained reasoning and allows to deduce knowledge missed by graph structure exploitation. Combining graph structure exploitation and propositional reasoning can give very good results for Maxclique solving, as shown in [14].

C. Combining graph structure exploitation and propositional reasoning

In propositional reasoning, unit propagation in a CNF formula refers to a procedure that repeatedly takes a unit clause l and satisfies it, i.e. assign the literal l to true, remove all clauses containing l because they are satisfied, and delete $\bar{l}$ from the other clauses. The procedure continues until no more unit clause exists in ϕ or an empty clause is produced (a clause becomes empty if it contained only $\bar{l}$). A literal l is failed if when it is assigned to true and $\bar{l}$ removed from all clauses, unit propagation produces an empty clause.

MaxCLQ uses the overestimation(G) function defined in Algorithm 3 to combine graph structure exploitation and propositional reasoning, allowing to improve the upper bound given by the independent subset partition P of G. The function

overestimation(G) repeatedly takes a soft clause c and tests if every literal in c is failed. If it is the case, the union of all the soft clauses used to produce an empty clause for every literal in c, together with c, constitute an inconsistent subset, since at least one soft clause in this subset cannot be satisfied by any assignment satisfying all the hard clauses.

Algorithm 3: overestimation(G), an overestimation of the cardinality of a maximum clique of G

```
Input: A graph G=(V, E)
Output: An upper bound for a maximum clique of G
1  begin
2      P ← partition(G);
3      Encode G into a MaxSAT formula φ based on P;
4      s ← 0;
5      while φ contains a non-tested soft clause do
6          c ← the soft clause of φ of the minimum size
           that is not tested;
7          mark c as tested;
8          if every literal l in c is a failed literal then
9              remove c and all the soft clauses making the
               literals of c failed from φ;
10             s ← s+1;
11     return |P| − s;
12 end
```

For example, the graph G in Figure 1 can be partitioned into three independent sets: $\{v_1, v_4, v_6\}$, $\{v_2, v_3\}$, $\{v_5\}$, giving an initial upper bound 3 for the cardinality of a maximum clique of G, which is the tightest upper bound that can be obtained by using a coloring process. Then the Maxclique instance can be encoded into a partial MaxSAT instance containing three soft clauses $\{x_1 \vee x_4 \vee x_6,\ x_2 \vee x_3,\ x_5\}$ and nine hard clauses: $\{\bar{x}_1 \vee \bar{x}_4,\ \bar{x}_1 \vee \bar{x}_5,\ \bar{x}_2 \vee \bar{x}_3,\ \bar{x}_2 \vee \bar{x}_5,\ \bar{x}_3 \vee \bar{x}_4,\ \bar{x}_1 \vee \bar{x}_6,\ \bar{x}_2 \vee \bar{x}_6,\ \bar{x}_4 \vee \bar{x}_6,\ \bar{x}_5 \vee \bar{x}_6\}$. Let us see that x_5 is a failed literal, and allows us to improve the upper bound. Set x_5=1 to satisfy the third soft clause, $\bar{x}_5$ is removed from the hard clauses $\bar{x}_1 \vee \bar{x}_5$, $\bar{x}_2 \vee \bar{x}_5$, and $\bar{x}_5 \vee \bar{x}_6$, the three new unit clauses imply that x_1=0, x_2=0 and x_6=0, then x_1 and x_6 are removed from the first soft clause, and x_2 from the second, which become unit. So, x_4 and x_3 should be assigned 1, making the hard clause $\bar{x}_3 \vee \bar{x}_4$ empty. So, x_5 is a failed literal and the three soft clauses used in the propagation to produce the empty clause constitute an inconsistent subset, because this subset, in conjunction with hard clauses, allows to derive a contradiction. Since at most two soft clauses can be satisfied by any assignment satisfying all the hard clauses, the function overestimation(G) improves the upper bound for a maximum clique from 3 to 2.

The function overestimation(G) detects s disjoint inconsistent subsets of soft clauses, and reduces the upper bound to $|P| - s$ using propositional reasoning, based on the fact that at least one soft clause in each inconsistent subset is not satisfied by any assignment satisfying all the hard clauses. In MaxCLQ, when propositional reasoning is called to reduce the upper bound, in up to 93% of cases, it succeeds in reducing the upper bound to the lower bound and prunes the subtrees rooted at the current node. Nevertheless, the improvement s is rather conservative, since several soft clauses in an inconsistent subset can be unsatisfied by an optimal assignment.

IV. Improvements to MaxCLQ

We present three improvement to the MaxSAT based approach presented in [14]. First, better graph structure exploitation as in Maxcliquedyn is implemented in MaxCLQ; then instead of detecting disjoint inconsistent subsets of soft clauses to reduce the upper bound, we use the soft clause relaxation approach proposed by Fu and Malik in [5] to underestimate the number of soft clauses unsatisfied in an optimal MaxSAT solution; finally we extend the failed literal detection used in MaxCLQ by the so-called *failed clause detection*. A clause is failed if all its literals are failed.

A. Dynamic re-computation of vertex degree in MaxCLQ

We adapt the dynamic re-computation of vertex degree of MaxCliquedyn in MaxCLQ, i.e., MaxCLQ re-computes vertex degree only at the search tree nodes of the lowest levels. Since the MaxSAT encoding at each search tree node of MaxCLQ depends on the independent set partition of the current graph G, which in turn depends on the degree information in G, the partition is presumably of worse quality and contains more independent sets at those search tree nodes where the re-computation is not performed. On the one hand, the behavior of propositional reasoning in MaxCLQ is changed and the success rate of propositional reasoning to prune subtrees might become worse. On the other hand, the fact that vertex degree is not recomputed in most nodes saves time. We conduct empirical investigation to determine the best value of the parameter T_{limit} in MaxCLQ. It appears that T_{limits}=0.075 gives better results than 0.025, the original best value in Maxcliquedyn, and several other values. In other words, MaxCLQ should re-compute vertex degree in three times more steps than Maxcliquedyn.

B. Soft clause relaxation in MaxCLQ

In [14], when MaxCLQ detects s disjoint inconsistent subsets of soft clauses, it reduces by s the upper bound given by an independent set partition of a graph G. This improvement is rather conservative, since it is just based on the fact that each inconsistent subset of soft clauses contains at least one unsatisfied soft clause by any assignment satisfying all the hard clauses, while there may be several soft clauses in an inconsistent subset unsatisfied by an optimal assignment. In other words, the minimum number of soft clauses unsatisfied by an optimal assignment can be bigger than s and the upper bound improvement can be more important. We want to use more powerful propositional reasoning to better underestimate the minimum number of soft clauses unsatisfied by an optimal assignment. For this purpose, we use the approach of Fu and Malik [5], which, instead of removing all the soft clauses in an inconsistent subset before detecting other inconsistent subsets of soft clauses, relaxes only one soft clause in the subset,

allowing to detect more inconsistent subsets of soft clauses in a partial MaxSAT instance.

Concretely, each time when Fu and Malik find an inconsistent subset of soft clauses in a partial MaxSAT instance ϕ using a modern SAT solver, say, $\{c_1, c_2, ..., c_t\}$, they add a new boolean variable y_i in each soft clause in the subset to form a longer clause $c_i \vee y_i$. Then they add a hard constraint $y_1 + y_2 + ... + y_t$=1 to say that exactly one variable y_i should be assigned to true, and the other new variables should be assigned to false. In this way, exactly one unsatisfied soft clause in the subset is relaxed and satisfied by assigning true to the corresponding variable y_i. The t longer clauses $c_i \vee y_i$, together with the hard constraint, are then used to detect other inconsistent subsets of soft clauses that are processed using the same procedure, until ϕ becomes satisfiable. The total number of inconsistent subsets of soft clauses found is the solution of ϕ, which is also the total number of relaxed soft clauses. This approach is implemented and improved by several authors in their MaxSAT solvers (see e.g., [15], [1]). In these solvers, the hard constraint $y_1 + y_2 + ... + y_t$=1 is encoded using a set of hard clauses.

In order to solve a Maxclique instance, we are interested only in inconsistent subsets of soft clauses detected using failed literal detection as described in Algorithme 3. The soft clauses in an inconsistent subset are no longer removed before detecting other inconsistent subsets. Instead, a new boolean variable is added in each soft clause in the subset together with the hard constraint $y_1 + y_2 + ... + y_t$=1. Since the longer soft clauses remain in the formula together with the hard constraint, more inconsistent subsets of clauses may be detected. Note that the number of inconsistent subsets detected is just an underestimation of the solution of the partial MaxSAT instance, since there are inconsistent subsets that cannot be detected using failed literal detection.

C. Extended failed literal detection in MaxCLQ

The length of a soft clause equals the number of vertices in an independent set in a MaxSAT instance encoding a Maxclique instance, if the encoding is based on an independent set partition of the graph. When the graph is not very dense, an independent set is generally relatively large, so that the corresponding soft clause presumably has low probability to become unit or empty during unit propagation. In reality, we observe that when a literal l is assigned to true, unit propagation produces often many binary soft clauses, among which there may be one whose two literals are all failed in the current context. In this case, l also can be considered as a failed literal, even if no empty clause is produced, since it implies a failed binary clause (i.e., if l is satisfied, then there is a soft clause that cannot be satisfied).

We extend the failed literal detection to capture this case. The extended failed literal detection is shown in Algorithm 4 and Algorithm 5.

The extended literal detection is used in the overstimation(G) function to detect if each literal of a soft clause c is failed. If it is the case, all the soft clauses making the literals of c failed constitute an inconsistent subset $\{c_1, c_2, ..., c_t\}$. Instead of being removed as presented in [14], a new boolean variable is added into each of these soft clauses together with a hard constraint $y_1 + y_2 + ... + y_t$=1, before detecting the next inconsistent subset.

Algorithm 4: failedLiteralDetection(ϕ, l), extended failed literal detection

Input: A partial MaxSAT instance ϕ not containing any empty clause and a literal l
Output: true if l is a failed literal, false otherwise

```
1  begin
2      l ← true;
3      Remove all clauses containing l;
4      Remove l̄ from all the other clauses;
5      φ ← unitPropagation(φ);
6      if φ contains an empty clause then
7          return true;
8      else
9          for each new binary soft clause c in φ do
10             if failedClauseDetection(φ, c)==true then
11                 return true;
12         return false;
13 end
```

Algorithm 5: failedClauseDetection(ϕ, c), Failed clause detection

Input: A partial MaxSAT instance ϕ not containing any empty clause and a clause c
Output: true if c is a failed clause, false otherwise

```
1  begin
2      for each literal l in c do
3          if failedLiteralDetection(φ, l)==false then
4              return false;
5      return true;
6  end
```

V. Experimental evaluation

We implement the different improvements presented in the last section in MaxCLQ and evaluate them on random graphs and DIMACS benchmarks, by comparing the different versions of MaxCLQ with the two best branch-and-bound algorithms MCR and Maxcliquedyn (MCQdyn for short) in our knowledge (apart from MaxCLQ). In [14], MCR and MCQdyn were shown substantially faster than Regin [19] and Cliquer [16]. In [20] and [19], MCR and Regin were respectively shown substantially faster than many other algorithms, including Fahler's algorithm [4], which in turn is improved from Carraghan and Pardalos's algorithm [2]. The famous DIMACS benchmark program dfmax is an implementation of Carraghan and Pardalos's algorithm. MCR and MCQdyn are also shown faster than MCQ in [8] and [20].

Table I and Table II compare the runtimes (in second) of MCR, MCQdyn and different versions of MaxCLQ for random graphs and DIMACS instances respectively. MCQdyn was obtained from one of its authors (D. Janezic) in Jan. 2010 and runs with the best parameter 0.025. MaxCLQ$^-$ uses Algorithm 2 to compute the upper bound for the cardinality of a maximum clique of a graph G and systematically recomputes vertex degree at every search tree node. The vertices with the maximum degree are inserted first into an independent set in Algorithm 2. With the exact degree information, the independent set partition is of better quality, but propositional reasoning is not used to improve the upper bound. We implement the heuristic dynamic recomputation of vertex degree of MCQdyn in MaxCLQ$^-$ to obtain MaxCLQdyn. In other words, MaxCLQdyn is the same as MCQdyn, except the implementation details and that it runs with the parameter 0.075 instead of 0.025 to recompute vertex degree at more search tree nodes.

MaxCLQdyn provides a basis for different improvements of MaxCLQ. MaxCLQdyn+FL implements the Failed Literal detection to detect disjoint inconsistent subsets of soft clauses, in order to improve the upper bound given by an independent subset partition of G in the function overestimation(G) as presented in [14] and shown in Algorithm 3. MaxCLQdyn+FL+SCR implements the Soft Clause Relaxation approach of Fu and Malik presented in Section IV-B. MaxCLQdyn+EFL+SCR implements the Extended Failed Literal detection together with the Soft Clause Relaxation approach as presented in Section IV-C. The runtimes of MCQdyn and all versions of MaxCLQ are obtained on a 3.33 Ghz intel core 2 duo CPU with linux and 4 Gb memory. The runtimes of MCR are normalized from the reported runtimes as follows.

The runtimes of the benchmark program dfmax for DIMACS graphs r100.5, r200.5, r300.5, r400.5, and r500.5 on our computer are respectively 0.004, 0.020, 0.172, 1.016, 3.872. The corresponding runtimes reported for the computer (Pentium4 2.20 GHz CPU with Linux) running MCR are 0.00213, 0.0635, 0.562, 3.48, 13.3. So, we divide the reported runtimes of MCR by 3.43 (=(13.3/3.872+3.48/1.016)/2=3.43, the average of the two largest ratios). This normalization is based on the way established in the Second DIMACS Implementation Challenge for Cliques, Coloring, and Satisfiability, and is also used in [20] and [14] to compare MCR and MaxCLQ with other algorithms.

In Table I, each algorithm except MCR solves 50 graphs at each point and the mean runtime is reported. The time cutoff is set to 3 hours. When there are instances that cannot be solved within 3 hours at a point, we give between brackets the number of instances solved and the mean time to solve these instances. MCQdyn is better than MaxCLQ$^-$ because of heuristic dynamic re-computation of the vertex degree. However, when propositional reasoning is used to improve the upper bound, MaxCLQdyn+FL is substantially better than MCQdyn for graphs with density ≥ 0.7, and the speed-up grows with density when the number of vertices is fixed, and with the number of vertices when the density is fixed. MaxCLQdyn+FL+SCR systematically improves MaxCLQdyn+FL, especially for dense graphes. Finally, the extended failed literal detection consistently and significantly improves MaxCLQdyn+FL+SCR, making MaxCLQdyn+EFL+SCR comparable with MCR even for sparse graphs (recall that in order to be efficient for sparse graphs, MCR does not recompute the vertex degree at all).

TABLE I

RUNTIMES IN SECOND FOR RANDOM GRAPHS, n STANDS FOR NUMBER OF VERTICES IN A GRAPH G, d FOR DENSITY, AND "-" MEANS THAT THE CORRESPONDING RUNTIME IS NOT AVAILABLE. TIME CUTOFF IS SET TO 3 HOURS. IF THERE ARE INSTANCES THAT CANNOT BE SOLVED WITHIN 3 HOURS AT A POINT, WE GIVE BETWEEN BRACKETS THE NUMBER OF INSTANCES SOLVED AND THE MEAN TIME TO SOLVE THESE INSTANCES.

G n	d	MCR	MCQdyn	MaxCLQ$^-$	MaxCLQdyn +FL	MaxCLQdyn +FL+SCR	MaxCLQdyn +EFL+SCR
200	0.50	**0.01**	**0.01**	0.03	0.02	**0.01**	**0.01**
200	0.60	**0.034**	0.05	0.18	0.08	0.05	0.05
200	0.70	0.27	0.28	0.63	0.31	0.24	**0.19**
200	0.80	5.14	3.20	6.37	2.02	1.75	**1.04**
200	0.90	285.8	55.92	84.66	16.81	8.65	**6.82**
200	0.95	-	51.10	76.38	7.16	2.36	**1.88**
300	0.50	**0.06**	0.09	0.26	0.19	0.16	0.12
300	0.60	**0.54**	0.64	1.46	0.79	0.78	0.72
300	0.70	9.00	8.23	19.22	7.54	7.01	**5.45**
300	0.80	521.9	280.8	536.7	162.7	125.3	**96.91**
300	0.90	-	- (0)	- (0)	9034(30)	4436	**3526**
300	0.99	-	- (0)	- (0)	0.10	0.07	**0.01**
400	0.50	-	**0.50**	1.92	1.12	0.96	0.66
400	0.60	-	**4.97**	10.86	7.12	6.10	5.22
400	0.70	-	111.1	261.6	99.63	91.72	**71.60**
400	0.80	-	8048(10)	- (0)	5899	3886	**3023**
400	0.99	-	- (0)	- (0)	9.48	0.92	**0.62**
500	0.50	**1.52**	2.00	4.78	4.05	3.10	2.36
500	0.60	**23.31**	27.02	74.82	35.74	31.63	30.23
500	0.70	1111	1018	2401	871.9	836.3	**649.6**
500	0.99	-	- (0)	- (0)	2593	214.6	**162.2**
600	0.50	-	**5.86**	13.71	11.84	6.12	6.01
600	0.60	-	127.6	312.6	162.6	129.8	**126.6**
600	0.70	-	7154(45)	- (0)	5939	5336	**4278**

In Table II, we exclude for the reason of clarity the very easy instances that are solved by all the algorithms in less than 1 second and the five open instances (MANN_a81, hamming10-4, johnson32-2-4, keller6, and p_hat1500-3) that no exact algorithm in our knowledge is able to solve. DIMACS instances generally have special properties. For example, the brock family graphs are generated by explicitly incorporating low-degree vertices into the optimal solution, and the p_hat family graphs have wider vertex degree spread and larger cliques than classical random graphs. We observe the similar results for DIMACS instances as in random graph case: MaxCLQdyn+FL is significantly faster than MCQdyn, MaxCLQdyn+FL+SCR significantly improves MaxCLQdyn+FL, and MaxCLQdyn+EFL+SCR improves MaxCLQdyn+FL+SCR, although this last improvement by extended failed literal detection is less important for DIMACS instances with special properties than for random graphs.

Note that the failed clause detection in MaxCLQdyn+EFL+SCR is rather time consuming, but we do not use any restriction in Algorithm 4 to select clauses when calling the failedClauseDetection(ϕ, c) function to see if a clause c

is failed. We believe that a heuristic based on an estimation of the probability for a clause to be failed can be used to restrict the clauses to test and might significantly further speed up MaxCLQdyn+EFL+SCR. Finally, observe that both MaxCLQdyn+FL+SCR and MaxCLQdyn+EFL+SCR are able to solve the open instance p_hat1000-3 in less than 30 hours on a 3.33Ghz core 2 duo CPU, while MaxCLQ presented in [14] needs more than 55 hours on a Macpro with a 2.8Ghz Xeon processor.

TABLE II
RUNTIMES IN SECOND FOR DIMACS INSTANCES, ω STANDS FOR THE OPTIMAL SOLUTION FOUND FOR AN INSTANCE. TIME CUTOFF IS SET TO 3 HOURS, EXCEPT FOR THE OPEN INSTANCE P_HAT1000-3

name	n	d	ω	MCR	MCQdyn	MaxCLQ$^-$	MaxCLQdyn +FL	MaxCLQdyn +FL+SCR	MaxCLQdyn +EFL+SCR
brock200_1	200	0.74	21	0.70	0.64	1.73	0.46	0.42	**0.37**
brock400_1	400	0.75	27	702.6	456.4	1235	307.9	266.5	**261.2**
brock400_2	400	0.75	29	287.5	201.4	570.6	134.9	118.6	**113.9**
brock400_3	400	0.75	31	473.8	366.1	823.3	269.8	232.6	**229.5**
brock400_4	400	0.75	33	253.1	208.2	510.6	124.9	110.2	**108.6**
brock800_1	800	0.65	23	6621	5994	>3h	5772	5351	**5338**
brock800_2	800	0.65	24	5982	5492	>3h	5192	4732	**4680**
brock800_3	800	0.65	25	4046	3749	>3h	3427	3106	**3004**
brock800_4	800	0.65	26	2819	2853	8047	2380	2185	**2178**
MANN_a27	378	0.99	126	1.22	1.52	3.12	0.79	**0.17**	**0.17**
MANN_a45	1035	0.996	345	1811	1216	10476	266.7	20.00	**19.28**
hamming10-2	1024	0.99	512	**0.10**	1.53	61.46	**0.10**	0.11	**0.10**
keller5	776	0.75	27	-	>3h	>3h	7909	6279	**6270**
p_hat300-3	300	0.74	36	4.61	2.94	8.56	1.62	1.45	**1.19**
p_hat500-2	500	0.50	36	1.31	0.98	3.18	0.67	0.51	**0.50**
p_hat500-3	500	0.75	50	776.4	206.3	539.5	72.46	**39.4**	45.91
p_hat700-2	700	0.50	44	18.66	7.86	20.55	4.22	3.12	**2.98**
p_hat700-3	700	0.75	62	-	3605	10398	1311	**862.1**	1040
p_hat1000-1	1000	0.24	10	**0.22**	0.36	2.39	1.12	0.78	0.34
p_hat1000-2	1000	0.49	46	1024	254.6	846	135.1	103.4	**99.22**
p_hat1000-3	1000	0.74	68	-	>48h	>48h	>48h	105881	**103698**
p_hat1500-1	1500	0.25	12	**1.84**	3.02	16.48	11.41	7.94	2.81
p_hat1500-2	1500	0.51	65	-	>3h	>3h	>3h	8088	7893
san1000	1000	0.50	15	2.07	0.48	1.35	0.59	**0.40**	**0.40**
san200_0.9_2	200	0.90	60	1.80	0.60	0.71	0.26	0.14	**0.08**
san200_0.9_3	200	0.90	44	**0.07**	2.02	5.53	0.39	0.21	0.18
san400_0.7_3	400	0.70	22	1.31	1.32	2.28	0.50	0.58	**0.48**
san400_0.9_1	400	0.90	100	1.55	18.13	50.33	2.09	0.97	**0.59**
sanr200_0.9	200	0.90	42	126.5	29.38	70.37	8.90	5.03	**4.74**
sanr400_0.5	400	0.50	13	**0.32**	0.50	1.71	1.07	0.78	0.43
sanr400_0.7	400	0.70	21	146.6	117.2	362.1	105.3	99.77	**89.39**

VI. CONCLUSION

Although general-purpose MaxSAT solvers are not competitive for solving Maxclique, propositional reasoning can be used to improve the upper bound obtained by partitioning a graph G into independent sets. We have proposed and studied several improvements to this approach, by integrating the heuristic dynamic re-computation of vertex degree of Maxcliquedyn, soft clause relaxation technique of Fu and Malik, and an extended failed literal detection allowing to detect more inconsistent subsets of soft clauses. Each improvement has been evaluated using random graphs and DIMACS instances. Experimental results show the effectiveness of the improvements.

In the future, we plan to improve the extended failed literal detection and integrate other effective MaxSAT technology into MaxCLQ to further improve its upper bound and to solve the harder Maxclique instances such as those in the BHOSLIB benchmark[2].

ACKNOWLEDGMENT

This work is supported by French ANR UNLOC project: ANR-08-BLAN-0289-03.

REFERENCES

[1] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. *Solving (weighted) partial MaxSAT through satisfiability testing*, In Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing, SAT-2009, Swansea, UK, pages 427–440. Springer LNCS 5584, 2009.

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

[3] M. Chudnovsky, N. Robertson, P. Seymour, R. Thomas, *The strong perfect graph theorem*. Annals of Mathematics 164 (1): 51-229, 2006

[4] T. Fahle, *Improving a branch-and-bound algorithm for maximum clique*, Proceedings of ESA-2002, pp. 485-498, 2002.

[5] Z. Fu, S. Malik, *On solving the partial Max-sat problem*, in proceedings of SAT 2006, LNCS 4121, pp 252-265, Springer, 2006

[6] M. Grtschel, L. Lovsz, A. Schrijver, (1988), *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag. See especially chapter 9, "Stable Sets in Graphs", pp. 273-303

[7] F. Heras, J. Larrosa, and A. Oliveras, *An efficient weighted Max-SAT solver*, Journal of Artificial Intelligence Research, 31:1-32, 2008.

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

[9] F. Heras, J. Larrosa, *A Max-SAT Inference-Based Pre-processing for Max-Clique*, in proceedings of SAT'2008, LNCS 4996, pp. 139-152, 2008.

[10] C. M. Li and F. Manyà, *Max-sat, hard and soft constraints. In A. Biere, H. van Maaren, and T. Walsh, editors*, Handbook of Satisfiability, Pages 613-631, IOS Press, 2009.

[11] C. M. Li, F. Manyà, and J. Planes, *New inference rules for Max-SAT*, Journal of Artificial Intelligence Research, 9(6): 375-382 (1990).

[12] C. M. Li, F. Manyà, and Jordi. Planes, *Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat*, Proceedings of AAAI'06, pages 86-91. AAAI Press, 2006.

[13] C. M. Li, F. Manyà and J. Planes, *Exploiting unit propagation to compute lower bounds in branch and bound MaxSAT solvers*, proceedings of CP'05, LNCS 3709 Springer, 2005, pp 403-414.

[14] C. M. Li, Zhe Quan, *An Efficient Branch-and-Bound Algorithm based on MaxSAT for the Maximum Clique Problem*, to appear in Proceedings of AAAI'2010, .

[15] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. *Algorithms for weighted boolean optimization*. In Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing, SAT-2009, Swansea, UK, pages 495–508. Springer LNCS 5584, 2009.

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

[17] P. R. J. Ostergard, *The maximum clique problem*, Journal of Global Optimization, 4: 301-328, 1994.

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

[19] J.-C. Regin, *Solving the maximum clique problem with constraint programming*, Proceedings of CPAIOR'03, Springer, LNCS 2883, pp. 634-648, 2003.

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

[21] E. Tomita, T. Seki, *An efficient branch-and-bound algorithm for finding a maximum clique*, Discrete Mathematics and Theoretical Computer Science, LNCS 2731, pp. 278-289 (2003).

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