An Efficient Branch-and-Bound Algorithm Based on MaxSAT for the Maximum Clique Problem

Chu-Min Li
Huazhong University of Science and Technology,
Wuhan, China
Université de Picardie Jules Verne, Amiens, France
http://www.laria.u-picardie.fr/˜cli/
chu-min.li@u-picardie.fr

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

Abstract

State-of-the-art branch-and-bound algorithms for the maximum clique problem (Maxclique) frequently use an upper bound based on a partition P of a graph into independent sets for a maximum clique of the graph, which cannot be very tight for imperfect graphs. In this paper we propose a new encoding from Maxclique into MaxSAT and use MaxSAT technology to improve the upper bound based on the partition P. In this way, the strength of specific algorithms for Maxclique in partitioning a graph and the strength of MaxSAT technology in propositional reasoning are naturally combined to solve Maxclique. Experimental results show that the approach is very effective on hard random graphs and on DIMACS Maxclique benchmarks, and allows to close an open DIMACS problem.

Keywords: Branch-and-Bound, Maxclique, MaxSAT

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 $\omega(G)$.

An independent set of G is a subset I of V such that no two vertices in I are connected. 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.

Maxclique is a very important NP-hard combinatorial problem, because it appears in many real-world applications. A huge amount of effort has been devoted to solve it. Pardolos and Xu (1994) gave 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. (Pullan & Hoos 2006), and exact algorithms including branch-and-bound algorithms, e.g. (Tomita & Kameda 2007; Konc & Janezic 2007; Ostergard 2002; Carraghan & Pardalos 1990; Fahle 2002; Regin 2003). 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 frequently use a heuristic solution to GCP as an upper bound. There are two drawbacks in this approach: (i) the number of independent sets in a partition of G generally is not minimum, since GCP 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. So, tighter upper bounds are needed in branch-and-bound algorithms for Maxclique.

There has recently been considerable progress in MaxSAT solving (see (Li & Manyà 2009) for a survey). 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 solve Maxclique, because their reasoning is not guided by the structural properties of the graph.

In this paper, we show that MaxSAT technology developed for MaxSAT solvers can be used to improve upper bounds in specific branch-and-bound algorithms for Maxclique. We first present some preliminaries and a basic branch-and-bound algorithm for Maxclique called MaxCLQ, before reviewing some previous successful upper bounds for Maxclique. Then we propose a new encoding from Maxclique into MaxSAT, which allows us to use MaxSAT technology to improve previous upper bounds for Maxclique. We then study the behaviour of our approach in MaxCLQ and compare MaxCLQ with the best state-of-the-art exact algorithms on random graphs and on the widely used DIMACS Maxclique benchmark[1]. The experimental results show that MaxCLQ is very efficient thanks to the integrated MaxSAT technology and, to our best knowledge, is able to solve an open DIMACS problem for the first time.

Preliminaries

A clique C of a graph G=(V, E) is maximum if no clique of larger cardinality exists in G. G can have several maximum cliques. Let V' be a subset of V, the subgraph G' of G induced by V' is defined as G'=(V', E'), where E'=$\{(v_i, v_j) \in E \mid v_i, v_j \in V'\}$. Given a vertex v of G, the set of neighbor vertices of v 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 $2m/(n(n-1))$.

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 problem into a MaxSAT problem 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 as a MaxSAT instance consisting of the soft clauses: $\{x_1, x_2, x_3, x_4, x_5, x_6\}$, and the hard clauses: $\{\bar{x}_1 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_5, \bar{x}_2 \vee \bar{x}_3, \bar{x}_2 \vee \bar{x}_5, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_6, \bar{x}_2 \vee \bar{x}_6, \bar{x}_4 \vee \bar{x}_6, \bar{x}_5 \vee \bar{x}_6\}$.

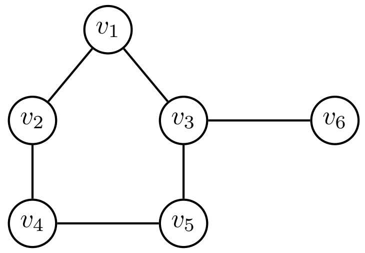

Figure 1: A simple imperfect graph ($\chi(G)$=3 and $\omega(G)$=2)

[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      $UB \leftarrow$ overestimation($G$)+$|C|$;
4      if $LB \geq UB$ then return $\emptyset$;
5      select a vertex $v$ from $G$ of the minimum degree;
6      $C_1 \leftarrow$ MaxCLQ($G_v$, $C \cup \{v\}$, $LB$);
7      if $|C_1| > LB$ then $LB \leftarrow |C_1|$ ;
8      $C_2 \leftarrow$ MaxCLQ($G \backslash v$, $C$, $LB$);
9      if $|C_1| \geq |C_2|$ then return $C_1$; else return $C_2$;
10 end
```

A Basic Branch-and-Bound Maxclique Solver

Algorithm 1 shows the pseudo-code of a branch-and-bound algorithm for Maxclique, inspired by the branch-and-bound algorithm MaxSatz for MaxSAT (Li et al. 2007). We illustrate the principle of this algorithm 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 a maximum clique in G which is clearly larger than 0. 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($G \backslash v_6$) uses the approach proposed in this paper, it will return an upper bound 2, 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.

Review of Previous Upper Bounds

Branch-and-bound algorithms for Maxclique frequently use heuristic solutions of GCP that can be obtained in reasonable time as their upper bound, based on the following property:

Proposition 1 *Let $\omega(G)$ denote the cardinality of a maximum clique of the graph G. If G can be partitioned into k* *independent sets, then $\omega(G) \leq k$.*

As a preprocessing, Algorithm Cliquer (Ostergard 2002) partitions G by determining an independent set at a time. As long as there are vertices that can be added into the independent set, one of these vertices with the largest degree is added. Then Cliquer constructs a maximum clique by incrementally considering vertices in the inverse order in which the vertices are added into independent sets, the upper bound for the largest cliques containing a certain vertex is quickly determined as a by-product.

Falhe (2002) improves the algorithm of Carraghan and Pardalos (1990), by using the constructive heuristic DSATUR to color vertices one by one, and by partitioning in parallel the graph into independent sets. As vertices are colored or inserted into an independent set in decreasing and increasing order of their degree respectively, four heuristic solutions of GCP are obtained, and the best one is used as the upper bound.

Regin (2003) uses an upper bound based on a matching algorithm. A matching, which corresponds to a set of independent sets of size 2 here, is computed by traversing the vertices of a graph and considering that an edge exists if two vertices are not connected.

MCQ (Tomita & Seki 2003) colors vertices 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 (Tomita & Kameda 2007) improves MCQ with a better initial order of vertices in the initial input graph, but uses the same coloring process to compute the upper bound.

MaxCliqueDyn (Konc & Janezic 2007) 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 some 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 coloring these vertices. The dynamic degree computation near the root of the search tree makes MaxCliqueDyn faster than MCQ for random graphs when their density is between 0.7–0.95, but slower than MCQ when the graph density is smaller than 0.7.

New Encodings from Maxclique into MaxSAT

Definition 1 *Let G be a graph partitioned into independent sets, the independent set based MaxSAT encoding of Maxclique is defined as follows: (1) each vertex v in G is represented by a Boolean variable x, (2) a hard clause $\bar{x}_i \vee \bar{x}_j$ is added for each pair of non-connected vertices (v_i, v_j), and (3) a soft clause is added for each independent set which is a logical or of the variables representing the vertices in the independent set.*

It is easy to see that the usual encoding from Maxclique into MaxSAT presented in Section preliminaries is a particular case of Definition 1, in which each independent set contains only one vertex.

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 hard clauses: $\{\bar{x}_1 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_5, \bar{x}_2 \vee \bar{x}_3, \bar{x}_2 \vee \bar{x}_5, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_6, \bar{x}_2 \vee \bar{x}_6, \bar{x}_4 \vee \bar{x}_6, \bar{x}_5 \vee \bar{x}_6\}$.

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, we have

Proposition 2 *Let ϕ be an independent set based MaxSAT encoding for a graph G, the set of variables evaluated to true in any optimal assignment of ϕ give a maximum clique of G.*

Different partitions of G into independent sets produce different MaxSAT instances. Table 1 compares three encodings on random graphs of 200 vertices and on some DIMACS graphs. The Max encoding is based on a partition computed using the coloring algorithm of MCQ to insert vertices one by one into an independent set in the decreasing order of their degree (i.e. vertices with the maximum degree are inserted first), Min is similar to Max except that vertices are inserted in the increasing order of their degree (i.e. vertices with the minimum degree are inserted first). We report, for each encoding, the number of independent sets in the graph, and the time needed to find a maximum clique by two state-of-the-art MaxSAT solvers MaxSatz (Li et al. 2007) and Minimaxsat (Heras et al. 2008). At each density of random graphs, 50 graphs are generated and encoded into MaxSAT. The average runtime and the average number of independent sets k are reported, k being also the number of soft clauses in the encoding. For both MaxSAT solvers, Max is the best performing encoding, the number of soft clauses being the smallest. We will use this encoding to apply MaxSAT technology in the next section.

Using MaxSAT Technology to Improve the Upper Bound for Maxclique

Given a partial MaxSAT instance, a branch-and-bound MaxSAT solver should find an assignment that minimizes the number of unsatisfied soft clauses and satisfies all the hard clauses. For this purpose, the solver should underestimate, at a search tree node, the number of soft clauses that will be unsatisfied by any complete assignment extending the current partial assignment. An approach proposed in (Li et al. 2005) and proved very powerful in MaxSatz and Minimaxsat consists in detecting disjoint inconsistent subsets of soft clauses. A subset of soft clauses is inconsistent if the subset, in conjunction with the set of hard clauses, allows to

Table 1: Run time [sec.] of Maxsatz (version 2009, MSZ in the table) and Minimaxsat (Mini in the table) on a Macpro 2.8 Ghz with 4Gb of memory for different MaxSAT encodings of Maxclique, k is the number of soft clauses in an encoding

$Graph$		$usual$			Min			Max		
name	density	k	MSZ	Mini	k	MSZ	Mini	k	MSZ	Mini
200	0.40	200	2.11	2.25	30.6	1.31	0.80	27.3	1.10	0.76
200	0.60	200	36.8	18.3	45.4	11.2	7.68	40.8	7.22	7.27
200	0.80	200	9691	576.3	67.4	265.7	278.5	61.1	117.8	269.2
brock200_1	0.74	200	1344	156.7	60	103.6	100.4	56	52.9	67.5
brock200_2	0.50	200	6.72	4.36	36	2.58	2.09	33	2.28	1.79
brock200_3	0.61	200	45.3	13.7	45	9.44	7.95	43	7.48	6.88
keller4	0.65	171	40.5	34.0	34	6.16	8.81	32	2.89	7.39
p_hat300-1	0.24	300	3.09	2.67	33	2.51	1.12	22	1.81	0.96
p_hat300-2	0.49	300	105.3	7.54	69	69.7	3.64	46	3.06	2.72
p_hat300-3	0.74	300	>3h	452.1	95	2326	163.5	72	185.3	129.2

derive a contradiction (i.e. an empty clause). The number of disjoint inconsistent subsets of soft clauses is a lower bound of the number of soft clauses unsatisfied by any complete assignment extending the current partial assignment and satisfying all the hard clauses. Observe that there is at least one unsatisfied soft clause in each inconsistent subset of soft clauses.

This approach, called UP, detects disjoint inconsistent subsets of clauses using propagation of unit clauses (unit propagation) as follows.

Given a CNF formula ϕ, UP stores all the unit clauses of ϕ in a queue Q, and applies unit propagation in a copy φ of ϕ, by repeatedly taking a unit clause l from Q to satisfy it (i.e. l=1). The satisfaction of l means that all clauses containing l are satisfied and removed from φ, and that $\bar{l}$ is removed from the other clauses, which may produce new unit clauses and empty clauses in φ (a clause becomes empty if it contained only $\bar{l}$). New unit clauses are stored at the end of Q. The unit propagation continues until there is no more unit clause in Q or an empty clause is produced. In the latter case, the clauses used in the propagation to produce the empty clause constitute an inconsistent subset of clauses. Once an inconsistent subset of clauses is identified, these clauses are removed from ϕ. UP continues to detect other inconsistent subsets of clauses in this way as long as there are unit clauses in ϕ. UP is enhanced in (Li et al. 2006) by failed literal detection. A literal l is failed if when it is satisfied and $\bar{l}$ removed from all clauses containing it, unit propagation produces an empty clause. If both l and $\bar{l}$ are failed literals, the union of the clauses used to produce the two empty clauses constitutes an inconsistent subset.

In a partial MaxSAT case, after an inconsistent subset of clauses is obtained using the above approaches, the hard clauses in the subset are not removed from ϕ and can be used to derive other contradictions, but the soft clauses in the subset are removed from ϕ. In other words, hard clauses can belong to several inconsistent subsets of clauses, because they have to be satisfied anyway. However, a soft clause cannot belong to two different inconsistent subsets of clauses.

We adapt failed literal detection to detect inconsistent subsets of soft clauses in a partial MaxSAT instance encoding a MaxClique instance. We do not detect whether a negative literal is failed or not, because a variable can have many negative occurrences but only one positive occurrence (in a soft clause). Instead, we detect if every literal in a soft clause is failed. In fact, given a soft clause c=$x_1 \vee x_2 \vee ... \vee x_t$, if every x_i ($1 \leq i \leq t$) is a failed literal, the union of all the soft clauses used to produce an empty clause for every x_i, together with c, constitute an inconsistent subset.

For example, in the independent set based MaxSAT encoding of the graph in Figure 1 presented in the last section, there are three soft clauses $\{x_1 \vee x_4 \vee x_6, x_2 \vee x_3, x_5\}$ corresponding to the three independent sets in an optimal partition of the graph. The hard clauses are: $\{\bar{x}_1 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_5, \bar{x}_2 \vee \bar{x}_3, \bar{x}_2 \vee \bar{x}_5, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_6, \bar{x}_2 \vee \bar{x}_6, \bar{x}_4 \vee \bar{x}_6, \bar{x}_5 \vee \bar{x}_6\}$. The initial upper bound for a maximum clique is 3, which is the tightest upper bound that can be obtained by using a coloring process. 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, we improve the upper bound for a maximum clique from 3 to 2.

In general, we have

Proposition 3 *Let $\omega(G)$ denote the cardinality of a maximum clique of a graph G. If G can be partitioned into k independent sets, and there are s disjoint inconsistent subsets of soft clauses in the independent set based MaxSAT encoding, then $\omega(G) \leq k - s$.*

Based on Proposition 3, Function overestimation(G) presented in Algorithm 2 first partitions G into independent sets in the same way as MCQ, except that the degree of each vertex in G is exact, so that the quality of the partition is presumably better (i.e., the partition presumably contains fewer independent sets), since vertices more constrained (i.e. with more neighbors) are inserted first into independent sets. Then the function encodes the graph into a MaxSAT instance based on the partition, which corresponds to the Max encoding presented in the last section, and uses effective MaxSAT technology to improve the upper bound given by the partition.

Using overestimation(G), the essential difference of MaxCLQ with a MaxSAT solver for Maxclique is as follows. While a MaxSAT solver uses a fixed MaxSAT encoding and never re-partitions the subgraphs during search, MaxCLQ dynamically partitions a subgraph and encodes the subgraph into MaxSAT to improve the upper bound given by the partition at every node of the search tree, so that the strength of the specific algorithms for Maxclique in partitioning a graph and the strength of the MaxSAT technology in propositional reasoning are naturally combined in MaxCLQ to solve Maxclique.

Algorithm 2: overestimation(G), an overestimation of the maximum cardinality of a clique of G

```
Input: A graph G=($V$, $E$)
Output: upper bound for a maximum clique of $G$
begin
    $P \leftarrow \emptyset$;
    while $G$ is not empty do
        $v \leftarrow$ the vertex of $G$ of the maximum degree;
        remove $v$ from $G$;
        if there is an independent set $S$ in $P$ in which $v$
        is not connected to any vertex then
            insert $v$ into $S$;
        else
            create a new independent set $S = \{v\}$;
            $P \leftarrow P \cup \{S\}$;
    Encode $G$ into a MaxSAT formula $\phi$ based on $P$;
    $s \leftarrow 0$;
    while $\phi$ contains a non-tested soft clause do
        $c \leftarrow$ the soft clause of $\phi$ of the minimum size
        that is not tested;
        mark $c$ as tested;
        if every literal $l$ in $c$ is a failed literal then
            remove $c$ and all the soft clauses making the
            literals of $c$ failed from $\phi$;
            $s \leftarrow s$+1;
    return $|P| - s$;
end
```

Comparative Evaluation of MaxCLQ

We compare the performance of MaxCLQ with MaxCLQ$^-$, which is identical to MaxCLQ except that it does not use MaxSAT technology to improve the upper bound in Function overestimation(G) (i.e., the function always returns $|P|$ in MaxCLQ$^-$), and the best exact algorithms for Maxclique in our knowledge: Cliquer, Algorithm Regin, MCR, MaxCliqueDyn (MCQdyn in short). The runtimes of MaxCLQ, MaxCLQ$^-$, Cliquer and MCQdyn are obtained on a Macpro with 2.8Ghz intel Xeon processor and 4 Gb memory (produced in early 2008) with MAC OS X 10.5. We use the last version of Cliquer released in 2008[2] and run it with its default parameters. MCQdyn was obtained from one of its authors (D. Janezic) in Jan. 2010 and run with the best parameter 0.025. The runtimes of MCR and Regin 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 the Macpro are respectively 0.002, 0.033, 0.270, 1.639, 6.281. 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 2.12 (=(13.3/6.281+3.48/1.639)/2, the average of the two largest ratios). The runtimes of dfmax for r100.5-r500.5 of Regin's computer (Pentium4 mobile 2Ghz with 512 Mb of memory) are not reported, but Regin's computer is about 10% slower than the computer running MCR, so we divide Regin's reported runtimes by 2.33. This normalization is based on the way established in the Second DIMACS Implementation Challenge for Cliques, Coloring, and Satisfiability, and is also used in (Tomita & Kameda 2007) to compare MCR with other algorithms. Regin (2003) normalizes the runtimes of algorithms by comparing different computers.

Table 2 compares the real runtimes of MaxCLQ, MaxCLQ$^-$, Cliquer, and MCQdyn with the normalized runtimes of MCR for random graphs up to 500 vertices. MaxCLQ, MaxCLQ$^-$, Cliquer, and MCQdyn solve 50 graphs at each point and the average runtime is reported. The graphs of low densities (e.g. graphs of 150 vertices and density 0.7) that are solved in less than 1 second by all the five algorithms are excluded to save space. MaxCLQ is substantially better than MaxCLQ$^-$ 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. MCR is faster than MaxCLQ for the relatively easy sparse graphs (density < 0.7), because it does not recompute the vertex degree at search tree nodes other than the root. However, MaxCLQ is substancially faster than MCR and other algorithms for graphs of density ≥ 0.7, and the speed-up also grows with the number of vertices and the graph density.

Table 2: Runtimes [sec] for random graphs. For Cliquer, MCQdyn and MaxCLQ$^-$, "-" means that an instance cannot be solved in 3 hours; for MCR, "-" means that the runtime is not available. The runtimes of Regin for random graphs are not available. s is the average upper bound improvement by MaxSAT, and Rate is the success rate of MaxSAT technology in MaxCLQ to prune subtrees (explained later), averaged for 50 graphs at each point.

n	density	Cliquer	MCR	MCQdyn	MaxCLQ$^-$	MaxCLQ	s	Rate
150	0.80	3.49	0.36	0.32	0.64	**0.16**	2.85	0.85
150	0.90	433.3	3.59	1.74	2.69	**0.25**	4.15	0.87
150	0.95	1513	2.01	0.74	1.17	**0.05**	2.68	0.54
200	0.70	2.06	**0.44**	0.47	1.06	**0.44**	2.35	0.82
200	0.80	125.0	8.32	5.12	10.12	**2.27**	3.19	0.87
200	0.90	-	462.4	90.73	138.3	**9.98**	4.87	0.91
200	0.95	-	-	81.4	121.9	**2.40**	6.10	0.90
300	0.60	3.27	**0.87**	1.02	2.32	1.49	2.18	0.74
300	0.70	112.1	14.57	12.12	28.25	**10.29**	2.81	0.82
300	0.80	-	844.3	423.8	805.9	**158.3**	3.39	0.88
300	0.90	-	-	-	-	**6695**	5.52	0.92
500	0.50	7.08	**2.13**	2.87	6.87	6.25	2.17	0.60
500	0.60	176.7	**37.71**	38.84	107.9	54.27	2.70	0.74
500	0.70	-	1797	1496	3527	**1147**	2.95	0.83

Table 3 compares the real runtimes of MaxCLQ, MaxCLQ$^-$, Cliquer, and MCQdyn with the normalized runtimes of MCR and Regin on DIMACS Maxclique benchmarks. In order to save space, we exclude the very easy instances that are solved by all the six algorithms in less than 2 seconds 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. Except 8 easy instances that MaxCLQ also solves quickly, MaxCLQ is significantly faster than all the other algorithms, especially for hard and dense graphs. Moreover, MaxCLQ is able to close the open instance p_hat1000-3.

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

Table 3: Runtimes [sec] for DIMACS benchmarks. "d" stands for the density. For Cliquer, MCQdyn and MaxCLQ$^-$, "-" means that an instance cannot be solved in 24 hours, except the instances keller5, p_hat1500-2 and p_hat1000-3 that cannot be solved in 5 days; for Regin and MCR, "-" means that the runtime is not available. s is the average upper bound improvement by MaxSAT, and Rate is the success rate of MaxSAT technology in MaxCLQ to prune subtrees (explained later).

name	n	d	ω	Cliquer	Regin	MCR	MCQdyn	MaxCLQ$^-$	MaxCLQ	s	Rate
brock200_1	200	0.74	21	6.37	4.60	1.13	0.96	2.28	**0.67**	2.66	0.86
brock400_1	400	0.75	27	22182	4867	1137	703.5	1447	**370.84**	2.92	0.86
brock400_2	400	0.75	29	5617	3395	465.10	309.0	664.9	**178.70**	2.86	0.86
brock400_3	400	0.75	31	1667	1922	766.51	565.0	971.3	**290.06**	2.81	0.85
brock400_4	400	0.75	33	247.7	2597	409.43	320.4	605.6	**167.30**	3.15	0.85
brock800_1	800	0.65	23	-	-	10712	8821	22821	**8815**	2.92	0.80
brock800_2	800	0.65	24	-	-	9679	8125	21001	**7690**	2.77	0.81
brock800_3	800	0.65	25	26014	-	6546	5565	13559	**5285**	2.73	0.80
brock800_4	800	0.65	26	6108	-	4561	4240	9625	**3880**	2.71	0.80
MANN_a27	378	0.99	126	-	7.93	1.98	3.10	6.86	**0.66**	3.21	0.75
MANN_a45	1035	0.996	345	-	-	2931	2006	8965	**255.67**	6.30	0.91
hamming10-2	1024	0.99	512	0.19	0.45	**0.16**	2.26	69.54	7.92	0.34	0.07
keller5	776	0.75	27	-	-	-	31038	78505	**9687**	3.60	0.87
p_hat300-3	300	0.74	36	496.6	17.47	7.45	4.91	9.79	**2.07**	3.07	0.85
p_hat500-2	500	0.50	36	134.6	14.03	2.12	1.53	3.92	**0.90**	3.26	0.83
p_hat500-3	500	0.75	50	-	5470	1256	349.4	634.4	**55.95**	4.56	0.91
p_hat700-1	700	0.25	11	0.09	2.58	**0.07**	0.14	0.71	0.80	1.88	0.33
p_hat700-2	700	0.50	44	15417	109.8	30.19	12.6	25.80	**4.87**	3.79	0.84
p_hat700-3	700	0.75	62	-	-	-	6187	12178	**1033**	4.64	0.91
p_hat1000-1	1000	0.24	10	1.11	11.93	**0.35**	0.58	2.84	2.53	1.21	0.77
p_hat1000-2	1000	0.49	46	-	7230	1656	412.9	1038	**146.54**	4.21	0.89
p_hat1000-3	1000	0.74	68	-	-	-	-	-	**200760**	5.45	0.93
p_hat1500-1	1500	0.25	12	8.01	206.4	**2.97**	4.31	22.37	15.85	2.19	0.82
p_hat1500-2	1500	0.51	65	-	-	-	61461	105909	**8848**	4.95	0.92
san1000	1000	0.50	15	**0.08**	44.12	3.35	0.74	1.56	1.46	1.67	0.61
san200_0.9_2	200	0.90	60	13.36	1.124	2.92	0.79	1.14	**0.10**	3.39	0.58
san200_0.9_3	200	0.90	44	503.4	78.41	**0.11**	3.43	5.80	0.22	4.86	0.82
san400_0.7_1	400	0.70	40	-	9.99	1.09	0.52	0.67	**0.21**	3.95	0.78
san400_0.7_2	400	0.70	30	3081	28.98	0.21	0.20	**0.09**	**0.09**	1.10	0.19
san400_0.7_3	400	0.70	22	4.47	117.3	2.12	2.04	2.67	**0.75**	2.66	0.83
san400_0.9_1	400	0.90	100	-	729.6	2.5	30.95	56.99	**1.97**	6.58	0.86
sanr200_0.7	200	0.70	18	1.88	1.845	**0.37**	0.39	0.86	0.38	2.40	0.80
sanr200_0.9	200	0.90	42	46593	64.41	204.72	51.57	80.51	**5.72**	5.07	0.91
sanr400_0.5	400	0.50	13	0.97	7.35	**0.52**	0.74	2.13	1.73	2.36	0.57
sanr400_0.7	400	0.70	21	2852	1347	237.26	177.8	429.25	**141.48**	2.52	0.85

In a search tree of MaxCLQ, let q denote the number of nodes in which MaxSAT technology sucessfully decreases the upper bound to the lower bound (i.e., $|P|+|C|>LB$, but $|P|-s+|C|\leq LB$) to prune the subtrees rooted at these nodes, and let r denote the number of nodes in which MaxSAT technology fails to decrease the upper bound to the lower bound (i.e., $|P|-s+|C|>LB$). In Table 2 and Table 3, in addition to runtimes, we report for MaxCLQ the average improvement s of the upper bound in these (q+r) nodes due to the MaxSAT technology, and the success rate q/(q+r) of the MaxSAT technology to prune subtrees. The success rate is quite high, especially for large and dense graphs (up to 93%), explaining the good performance of MaxCLQ.

Conclusion

We have proposed a new encoding from Maxclique into MaxSAT based on a partition of a graph into independent sets, each independent set being encoded into a soft clause. The number of soft clauses is substantially smaller than in the usual MaxSAT encoding of Maxclique. The new encoding allows us to naturally integrate MaxSAT technology into a branch-and-bound algorithm to improve the upper bound based on the partition. Experimental results show that many subtrees are pruned in this way and the resulting algorithm MaxCLQ is very efficient, especially for hard instances, and is able to close an open DIMACS instance.

We believe that using MaxSAT technology to improve the upper bound for Maxclique is a very promising research direction. In the future, we plan to 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[3].

References

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

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

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

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.

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.

C. M. Li, F. Manyà, and J. Planes, New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321-359, 2007.

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

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

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

P. M. Pardalos, J. Xue, The maximum clique problem. *Journal of Global Optimization* 4: 301-328, 1994

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

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

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.

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

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