Using Constraint Programming to Solve the Maximum Clique Problem

Jean-Charles Régin

ILOG Sophia Antipolis
Les Taissounières HB2,
1681 route des Dolines,
06560 Valbonne, France
`regin@ilog.fr`

Abstract. This paper aims to show that Constraint Programming can be an efficient technique to solve a well-known combinatorial optimization problem: the search for a maximum clique in a graph. A clique of a graph $G = (X, E)$ is a subset V of X, such that every two nodes in V are joined by an edge of E. The maximum clique problem consists of finding $\omega(G)$ the largest cardinality of a clique. We propose two new upper bounds of $\omega(G)$ and a new strategy to guide the search for an optimal solution. The interest of our approach is emphasized by the results we obtain for the DIMACS Benchmarks. Seven instances are solved for the first time and two better lower bounds for problems remaining open are found. Moreover, we show that the CP method we propose gives good results and quickly.

Introduction

Constraint Programming (CP) involves finding values for problem variables subject to constraints on which combinations are acceptable. One of the main principles of CP is that every constraint is associated with a filtering algorithm (also called a domain reduction algorithm) that removes some values that are inconsistent with the constraint. Then, the consequences of these deletions are studied thanks to a propagation mechanism that calls the filtering algorithms of the constraints until no more modification occurs. CP uses also a systematic search, like a branch-and-bound for instance, but this is not limited to this case, to find solutions.

In this paper, we aim to contradict some conventional wisdom of Constraint Programming. It is often considered that CP is not an efficient method to solve pure combinatorial optimization problems. By "pure problems", we mean problems in which only one kind of constraint is involved.

A clique of a graph $G = (X, E)$ is a subset V of X, such that every two nodes in V are joined by an edge of E. The maximum clique problem consists of finding $\omega(G)$ the largest cardinality of a clique. Finding a clique of size k is an NP-Hard problem. This problem is quite important because it appears in a lot of real world problems. Therefore almost all types of algorithms have been used to try to solve it. For more information the reader can consult the survey of Bomze, Budinich, Pardalos and Pelillo [3].

Fahle [7] has proposed to use CP techniques to solve this problem. The results he obtained were encouraging. Notably, he has been able to close some open problems. His model uses two constraints: one based on the degree, we will call it degreeCt, and one based on the search for an upper bound of the size of a maximum clique, we will call it UBMaxCliqueCt. These two constraints are defined on the set of nodes that are considered at every moment by the algorithm. Then a branch-and-bound algorithm is used to traverse the search space.

Fahle's algorithm tries to construct a clique as large as possible, by successively selecting a node and studying the **candidate set**, that is the set of nodes that can extend the clique currently under construction. After each selection of node, the filtering algorithms associated with the two constraints are triggered until no more modification of the candidate set occurs.

The filtering algorithm associated with degreeCT removes all nodes whose degree is too small to extend the current clique to a clique of size greater than the current objective value.

The filtering algorithm associated with UBMaxCliqueCt removes all nodes for which we know that they cannot belong to a clique of size greater than the current objective value. A non obvious bound is searched by computing an upper bound of the number of colors needed to color the subgraph induced by a node and its neighborhood such that two adjacent nodes have different colors.

The drawback of this filtering is the time required to compute such a bound, and also its systematic use. That is, a priori, we do not know whether the filtering algorithm will remove some values or not.

In this paper we propose to use another upper bound based on matching algorithm. The advantage of our method is that we can easily identify some cases for which the filtering algorithm will remove no value; and so we can avoid to call it.

Moreover, Fahle uses a common strategy to select the next node that will extend the current clique under construction. This strategy is based on the degree of the nodes and selects the one with the smallest value. We propose a different approach that can be viewed as an adaptation and a generalization of the Bron & Kerbosh's [4] ideas for enumerating the maximal cliques of a graph. This idea leads to a new filtering algorithm based on the study of the nodes that have already been tried. Our strategy is more complex but tends to find more quickly the cliques with a large size as it is shown by the results we obtain on the well-known DIMACS benchmarks.

The paper is organized as follows. First we present, new upper bounds for the maximum clique problem. Then, we introduce some new properties that are based on the ideas of the Bron & Kerbosh's algorithm. The strategy that exploits the previous ideas is detailed. After, we will give some results. Finally, we present some ideas about the definition of a maximum clique constraint and we conclude.

1 Preliminaries

1.1 Graph

A **graph** $G = (X, E)$ consists of a **node set** X and an **edge set** E, where every edge (u, v) is a pair of distinct nodes. u and v are the endpoints of (u, v). The **c**omplementary graph of a graph $G = (X, E)$ is the graph $\overline{G} = (X, F)$, where (x, y) is an edge of $\overline{G}$ if and only if $x \neq y$ and (x, y) is not an edge of G.

A **path** from node v_1 to node v_k in G is a list of nodes $[v_1, ..., v_k]$ such that (v_i, v_{i+1}) is an edge for $i \in [1..k-1]$. The path **contains** node v_i for $i \in [1..k]$ and arc (v_i, v_{i+1}) for $i \in [1..k-1]$. The path is a **cycle** if $k > 1$ and $v_1 = v_k$. The **length** of a path p, denoted by $length(p)$, is the number of arc it contains. $\Gamma(x)$ is the set of neighbors of x, that is the set of nodes y such that $(x, y) \in E$.

A **clique** of a graph $G = (X, E)$ is a subset V of X, such that every two nodes in V are joined by an edge of E. The **maximum clique problem** consist of finding $\omega(G)$ the largest cardinality of a clique. Given a node x, $\omega(G, x)$ denotes the size of the largest clique containing x.

A **independent set** of a graph $G = (X, E)$ is a subset S of X, such that every two nodes in V are not joined by an edge of E. The **maximum independent set** problem consist of finding $\alpha(G)$ the largest cardinality of an independent set.

A **vertex cover** of a graph $G = (X, E)$ is a subset V of X, such that every edge of E has an endpoint in V. The **minimum vertex cover** problem consist of finding $\nu(G)$ the smallest cardinality of a vertex cover.

A **matching** of a graph $G = (X, E)$ is a subset M of E, such that no two edges of M have a common node. The **maximum matching** problem consists of finding $\mu(G)$ the largest cardinality of a matching.

1.2 CP Algorithm for solving maximum clique

Let $G = (X, E)$ be a graph. The idea is to start with a clique $C = \emptyset$, called the **current set**, and a **candidate set** equals to X. Then the algorithm successively selects nodes in the candidate set in order to increase the size of C. When a node x is added to C, all the nodes that are non adjacent to x are removed from the candidate set. The candidate set is also used for bounding. Algorithm 1 is a possible implementation. The algorithm must be called with $Current = \emptyset$, $Candidate = X$ and $K = \emptyset$, where K is the largest clique found so far. Function filterAndPropagate returns false when we can prove that there is no clique whose cardinality is strictly greater than $|K|$ in the subgraph of G induced by $(Current \cup Candidate)$; otherwise it returns true. This function also aims to remove some values of $Candidate$ that cannot belong to a clique of size strictly greater than $|K|$ and containing $Current$. The simplest condition to remove a node y is to check whether $|\Gamma(y) \cap Candidate| + |Current| < |K|$

Every upper bound for the maximum clique problem is interesting in a CP approach, because we will use it to check whether a node can belong to a clique of a given size.

Algorithm 1: Basic Algorithm for searching for a maximum Clique

```
maximumClique($Current, Candidate,$ **io** $K$)
while $Candidate \neq \emptyset$ do
    select $x$ in Candidate and remove it
    save $Candidate$
    add $x$ to $Current$
    remove from Candidate the nodes $y$ s.t. $y \notin \Gamma(x)$
    if filterAndPropagate($Current, Candidate, K$) then
        if $Candidate = \emptyset$ then $K \leftarrow Current$ // solution
        else maximumClique($Current, Candidate, K$)
    remove $x$ from $Current$
    restore $Candidate$
```

Property 1 *Let $G = (X, E)$ be a graph and x a node, and K be a clique of G. If $\omega(G, x) < |K|$ then $\omega(G) = \omega(G - \{x\})$.*

Therefore any upper bound of $\omega(G, x)$ can be used to remove some nodes in the candidate set. A simple bound can be $|\Gamma(x) \cap Candidate| + |Current|$. That is, as proposed by [7], we can remove from the candidate set all the nodes such that $|\Gamma(x) \cap Candidate| + |Current| < |K|$. The deletion of a node modifies the neighborhood of its neighbors thus it can change the value of the upper bound of some other nodes, so the process is repeated until no more modifications occurs. Function filterAndPropagate given by Algorithm 2 implements this idea.

Algorithm 2: Filtering algorithm and propagation

```
filterAndPropagate($Current, Candidate,$ **io** $K$)
do
    $continue \leftarrow false$
    for each $y$ in $Candidate$ do
        if $|\Gamma(y) \cap Candidate| + |Current| < |K|$ then
            remove $y$ from $Candidate$
            if $|Candidate| + |Current| < |K|$ then return false
            $continue \leftarrow true$
while $continue$
return true
```

2 Upper bounds for clique

If we find better upper bounds for the size of the maximum clique involving a node then we will be able to improve function filterAndPropagate and so to remove more values.

There are some relations between the maximum clique problem, the maximum independent set, the minimum vertex cover, and the maximum matching:

Property 2 *Let $G = (X, E)$ be a graph, then*
- $\omega(G) = \alpha(\overline{G})$
- $\alpha(G) = |X| - \nu(G)$
- $\omega(G) = |X| - \nu(\overline{G})$

proof: A maximum clique corresponds to an independent set in the complementary graph, hence $\omega(G) = \alpha(\overline{G})$. The subgraph induced by an independent set S does not contain any edge, thus every edge of G has an endpoint in $Y = X - S$, therefore Y is a vertex cover of G. Hence $S = X - Y$ and the largest set S is associated with the smallest set Y, so $\alpha(G) = |X| - \nu(G)$. Then, $\omega(G) = |X| - \nu(\overline{G})$ follows immediately.

Property 3 *Let $G = (X, E)$ be a graph, then*
$\nu(G) \geq \mu(G)$ and the equality holds if G is bipartite.

proof: Let V be any vertex cover of G. All the edges of a matching have no common nodes, thus at least one endpoint of every edge of the matching must be in V in order to cover this edge. Therefore $\nu(G) \geq \mu(G)$. The proof for the bipartite case can be found in [2].

From this property and the previous one we immediately deduce the well known property:

Property 4 $\omega(G) \leq |X| - \mu(\overline{G})$

This new upper bound could be used, but it has one drawback: $\overline{G}$ can be non-bipartite, and the algorithm to compute a maximum matching in a non-bipartite graph is complex. Thus we propose to use an original upper bound for $\omega(G)$, which is stronger and much more easy to compute. We need, first, to define the duplicated graph of a graph (See Figure 1.)

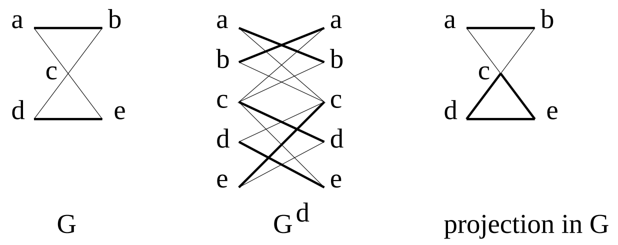

Fig. 1. An example of a duplicated graph of a graph. The bold edges represent the edges of the matchings. The right graph is the projection of the matching of G^d in G. G is covered by an edge and a triangle, therefore $1 + 2$ nodes are necessary to cover all the edges and $\nu(G) \geq 3$.

Definition 1 *Let $G = (X, E)$ be a graph. The* **duplicated graph** *of G is the bipartite graph $G^d = (X, Y, F)$, such that Y is a copy of the nodes X, $c(u)$ is the* *node of Y corresponding to the node u in X, and there is an edge $(u, c(v))$ in F if and only if there is an edge (u, v) in E.*

Note that if $(u, v) \in E$ then $(v, u) \in E$.

Property 5 *$\mu(G^d) \geq 2.\mu(G)$ and there exist graphs G with $\mu(G^d) > 2.\mu(G)$*

proof: From a matching M of G we can create a set M' of edges of G^d as follows: for each edge (u, v) in M we add the edges $(u, c(v))$ and $(v, c(u))$ to M'. M is matching thus it involves $2.|M|$ nodes. By construction of M' and by definition of G^d, M' involves $2.|M|$ nodes of X and $2.|M|$ nodes of Y. Therefore M' is matching of size $2.|M|$ and $\mu(G^d) \geq 2.\mu(G)$. A triangle is an example of graph G with $\mu(G^d) = 3$ and $\mu(G) = 1$, that is $\mu(G^d) > 2.\mu(G)$ (See also Figure 1.)

We define the projection of a matching of G^d in G:

Definition 2 *Let $G = (X, E)$ be a graph and M be a matching of G^d. Let E' by the subset of E defined by $(u, v) \in E'$ if and only if either $(u, c(v))$ or $(v, c(u))$ belongs to M.*
The **projection** *of M in G is the subgraph of G induced by the subset E' of E. We will denote it by $P(M, G)$.*

Figure 1 contains an example of projection.

We will denote by $edges(cc)$ the edge set of a connected component cc.

Property 6 *Let $G = (X, E)$ be a graph, M be a matching in G^d, $P(M, G)$ be the projection of M in G, and CC be the set of the connected components of $P(M, G)$. Then*

$$\nu(G) \geq \sum_{cc \in CC} \lceil \frac{|edges(cc)|}{2} \rceil$$

proof: Consider cc a connected component of $P(M, G)$. M is matching, so no node of $P(M, G)$ can have a degree greater than 2. Therefore, cc is either an isolated node, or a path, or a cycle. Hence, the number of nodes needed to cover the edges of cc is $\lceil \frac{|edges(cc)|}{2} \rceil$. The connected components are node disjoint therefore the value associated with each component can be sum, and the property holds.

From this property we can also deduce a simpler property which is weaker but interesting due to property 5.

Property 7 $\nu(G) \geq \lceil \frac{\mu(G^d)}{2} \rceil$

proof: Consider a matching M in G^d with $M = \mu(G^d)$, $P(M, G)$ the projection of M in G, and cc a connected component of $P(M, G)$. As mentioned in the proof of Property 6, cc is either an isolated node, or a path, or a cycle. Let $Medges(cc)$ be the set of edges $(u, c(v))$ of M such that (u, v) belongs to cc. If cc is a path containing only one edge then $|Medges(cc)| = 2$ and $\lceil \frac{|edges(cc)|}{2} \rceil = \lceil \frac{|Medges(cc)|}{2} \rceil$. In all the other cases $|edges(cc)| = |Medges(cc)|$, and so $\lceil \frac{|edges(cc)|}{2} \rceil = \lceil \frac{|Medges(cc)|}{2} \rceil$. Moreover, by definition of the projection of M, for every edge $(u, c(v))$ of M there exists a connected component cc containing (u, v), then we have $\sum_{cc \in CC} \lceil \frac{|Medges(cc)|}{2} \rceil = \sum_{cc \in CC} \lceil \frac{|edges(cc)|}{2} \rceil$.

On the other hand $\sum \lceil \frac{i}{2} \rceil \geq \lceil \frac{\sum i}{2} \rceil$
thus we have:

$$\sum_{cc \in CC} \lceil \frac{|edges(cc)|}{2} \rceil = \sum_{cc \in CC} \lceil \frac{|Medges(cc)|}{2} \rceil \geq \lceil \frac{\sum_{cc \in CC} |Medges(cc)|}{2} \rceil \ \geq \lceil \frac{\mu(G^d)}{2} \rceil$$

And from Property 6 we deduce: $\nu(G) \geq \lceil \frac{\mu(G^d)}{2} \rceil$

Finally, from Property 2 we immediately have the property:

Property 8 $\omega(G) \leq |X| - \lceil \frac{\mu(\overline{G}^d)}{2} \rceil$

We decided to base our filtering algorithm on this property and not on Property 6 because with Property 8 we have a good test to know whether it can be interesting to check it. We just have to check whether $|X| - \lceil \frac{|X|}{2} \rceil$ is lower than the size of the clique currently computed. From our experiments, thanks to this test, only 5% of the matching that are computed are useless[1].

We have also implemented Property 6 but the gain is really small in term of eliminated nodes and more time is needed. We would like to stress on this point. CP involves a propagation mechanism, therefore, and especially for pure problems, the comparison of two properties is not simple because we have to take into account the propagation mechanism. Here, we have seen that the use of Property 8 and propagation gives equivalent result to the use of Property 6 with propagation, so we can eliminate the use of the second, if, of course, it required more time to be checked. Moreover, in practice, we can stop the computation of the maximum matching either when the current size of the matching is enough to conclude that we cannot find a clique with a largest size of the better found so far, or when we known that we will not be able to make such a deduction.

On the other hand, in practice there is an important difference between Property 4 and Property 8. It seems really worthwhile to improve the upper bound even by a small value, provided that we have an interesting test to avoid some useless computations.

Function FILTERANDPROPAGATE can be refined. Algorithm 3 gives its new code.

There is no need to explicitly create the subgraph in the function it is sufficient to traverse the nodes of $\Gamma(x) \cap Candidate$ and the matching can be computed by considering that an edge exists if two nodes are non adjacent. In the algorithm, we also apply this new test for the set *Candidate* when a node is removed in order to know whether it is useless to continue the search.

3 Introduction of a not set

When enumerating all the maximal cliques of a graph, Bron and Kerbosh [4] have proposed to use a new set of nodes: a **not set** denoted by *Not*. This set

[1] More precisely, only 5% of nodes that satisfies this test (that is mark 1 in Algorithm 3) will not be removed by Algorithm 3 (that is by mark 2.)

Algorithm 3: Filtering algorithm and propagation: a new version

```
    filterAndPropagate($Current, Candidate,$ io $K$)
    do
        $continue \leftarrow false$
        for $each\ y\ in\ Candidate$ do
            $N \leftarrow |\Gamma(y) \cap Candidate|$
            if $N + |Current| < |K|$ then remove $y$ from $Candidate$
            else
1               if $N - \lceil \frac{N}{2} \rceil + |Current| < |K|$ then
                    Let $H$ be the subgraph of $G$ induced by $\Gamma(x) \cap Candidate$
                    compute $\mu(\overline{H}^d)$
                    if $N - \lceil \frac{\mu(\overline{H}^d)}{2} \rceil + |Current| < |K|$ then
2                       remove $y$ from $Candidate$
            if $y \notin Candidate$ then
                Let $H$ be the subgraph of $G$ induced by $Candidate$
                compute $\mu(\overline{H}^d)$
                if $|Candidate| - \lceil \frac{\mu(\overline{H}^d)}{2} \rceil + |Current| < |K|$ then return false;
                $continue \leftarrow true$
    while $continue$
    return true
```

contains the nodes that have already been studied by the algorithm and that are linked to all the nodes of the *Current* set. We propose to adapt their idea to our case and to generalize it.

In order to clearly understand the meaning of the not set we propose to immediately adapt our algorithm (See Algorithm 4.) Function removeFromNot(x) removes from *Not* the element that are not in $\Gamma(x)$.

The idea of Bron and Kerbosh corresponds to the following property:

Property 9 *If there is a node x in Not such that $Candidate \subseteq \Gamma(x)$ then the current branch of the search can be abandoned.*

proof: all cliques that we can find from the current set and from the candidate set will be a clique by adding x, therefore these cliques cannot be maximal.

This property can be refined when searching for the size of a maximum clique. A dominance property can be obtained:

Dominance Property 1 *If there is a node x in Not such that $|Candidate - \Gamma(x)| \leq 1$ then the current branch of the search can be abandoned.*

proof: We just have to consider the case $Candidate - \Gamma(x) = \{y\}$. There are two possible cliques: the cliques that contain y and the cliques that do not contain y. Consider any clique that does not contain y. In this case, this clique could also be found if y is removed from *Candidate* and therefore Property 9 can be applied. Consider any clique that contains y, then if we replace y by x we also obtain a clique because x is linked

Algorithm 4: Solving the maximum Clique problem: introduction of a not set

```
maximumClique($Current, Candidate, Not,$ **io** $K$)
while $Candidate \neq \emptyset$ do
    select $x$ in Candidate and remove it
    save $Candidate$
    save $Not$
    add $x$ to $Current$
    remove from Candidate the nodes $y$ s.t. $y \notin \Gamma(x)$
    removeFromNot($x$)
    if filterAndPropagate($Current, Candidate, Not, K$) then
        if $Candidate = \emptyset$ then $K \leftarrow Current$ // solution
        else maximumClique($Current, Candidate, K$)
    restore $Not$
    remove $x$ from $Current$
    add $x$ to $Not$
    restore $Candidate$
```

to all the nodes except y, and the size of the clique is unchanged. And, this clique has already been found when x has been selected, hence we cannot improved the largest cardinality found so far.

This new property leads to a modification of our algorithm. In the Bron and Kerbosh's algorithm a node is removed from Not when the selected node is not linked to it. In our case, we slightly change this property: instead of removing a node x in Not when a selected node y is not linked to it, we can mark x if it is unmarked and remove x if it is already marked. Our property must be changed:

Dominance Property 2 *If there is a node x in Not such that x is not marked and $|Candidate - \Gamma(x)| \leq 1$ then the current branch of the search can be abandoned.*
If there is a node x in Not such that x is marked and $Candidate \subseteq \Gamma(x)$ then the current branch of the search can be abandoned.

Function removeFromNot has to be accordingly modified. From this property we can define a new filtering algorithm:

Unfortunately, the cost of checking this property is high. In practice, it is not worthwhile to use it. We have preferred to use it in a different way. Instead of searching if the current neighborhood of every node of the candidate set is included in the neighborhood of every node in Not, we decided to limit our study to the node of not whose neighborhood contains almost all nodes of candidate. That is, for every node x of Not we compute the number of nodes of candidates that are linked to x. If this number is greater than $|Candidate| - 2$ we can immediately identify the nodes of the candidate set that must be selected or that must be removed. The strict application of Dominance Property 2 gives better results in terms of backtracks (around 10% less) than the restriction we propose. However, the latter approach is much more easy to implement and more efficient to compute, because we only need to compare the neighborhood of a node of *Not* with the candidate set and not with neighborhood of every node in the candidate set taken separately.

The final version of our algorithm is given by Algorithm 5.
Function FILTERINGFROMNOT uses the notion of mark or removes some nodes.

Algorithm 5: Filtering algorithm and propagation taken into account nodes of the not set

```
FILTERANDPROPAGATE($Current, Candidate,$ io $K$)
do
    do
        $continue \leftarrow false$
        for each $y$ in $Candidate$ do
            $N \leftarrow |\Gamma(y) \cap Candidate|$
            if $N + |Current| < |K|$ then remove $y$ from $Candidate$
            else
                if $N - \lceil \frac{N}{2} \rceil + |Current| < |K|$ then
                    Let $H$ be the subgraph of $G$ induced by $\Gamma(x) \cap Candidate$
                    compute $\mu(\overline{H}^d)$
                    if $N - \lceil \frac{\mu(\overline{H}^d)}{2} \rceil + |Current| < |K|$ then remove $y$ from $Candidate$
                if $y \notin Candidate$ then
                    Let $H$ be the subgraph of $G$ induced by $Candidate$
                    compute $\mu(\overline{H}^d)$
                    if $|Candidate| - \lceil \frac{\mu(\overline{H}^d)}{2} \rceil + |Current| < |K|$ then return false; $continue \leftarrow true$
    while $continue$
    if $continue$ then
        $continue \leftarrow$ FILTERINGFROMNOT($Not, Candidate$)
while $continue$
return true
```

4 A new search strategy

As it has been shown by Bron and Kerbosh to enumerate the maximal cliques of a graph, it is interesting to select node such that Property 9 can be applied as soon as possible.

This means that when a node is added to *Not*, we identify first the node x in *Not* which has the largest number of neighbors in the candidate set. Then, we select for next node, a node y such that y is not linked to x

We have used exactly the same idea by considering in *Not* only the unmarked nodes. The ties have been broken by selecting the node y with the fewest number of neighbors in the candidate set, in order to have more chance to remove quickly y and then to be able to apply successfully Dominance Property 2.

When a node has not been removed, that is when the latest selection is successful; we select the node with the largest number of neighbors in the candidate set. This approach gives a better chance to find quickly cliques whose cardinality is huge. This is important for our approach because our filtering algorithm takes into account the size of the clique found so far.

4.1 Diving technique

This technique is often used in conjunction with a MIP approach. It consists of searching whether of solution exists for every value of every variable. Each search for a solution is not complete. In other words, a greedy algorithm is used (that is no backtrack is allowed). Then, the new objective value is the best objective value found so far. The advantages of this approach is triple: its cost is low because the algorithm is polynomial, the minimum of the objective value can be improved, and an objective value can be quickly found whereas a depth first search strategy will need a lot of time to find it. In fact, a systematic search spends a lot of time to proved the local optimality of the current objective value; this proof is abandoned when a better value is found.

In our program, this technique is used after 10 minutes of computations. That is, we stop the current search, we apply the diving technique and the initial search continues with the objective value returned by the diving technique, which can be improved or not.

5 Experiments

We have used ILOG Solver to implement our algorithm and the well known DIMACS benchmark set for our tests [6].

All our experiments have been made on a Pentium IV mobile at 2Ghz with 512 Mo of memory.

The experiments have been stopped after 4 hours (that is 14,400 s) of computation, except for p_hat1000-2 because we saw after 14,400 s that the problem should be solved. In this case, 16,845 s are needed to close the problem.

The results are given in table entitled "Dimacs clique benchmarks".

All the problems having 400 nodes or less are solved. Notably, for the first time the brock400 series is now solved. Only, johnson32-2-4, prevent us from solving all the problems having 500 nodes or less.

All san series or sanr series are now solved.

7 problems have been closed for the first time: all the brock400 series, p_hat500-3, p_hat1000-2 and sanr200_0.9, which is solved in 150 s.

Two results are particularly remarkable: p_hat300-3 is solved in 40s, instead of 850s; and p_hat700-2 is solved in 255s instead of 2,086s.

Two lower bounds of the remaining open problems MANN_a45 (the optimal value is reached) and MANN_a81 have been improved.

DIMACS CLIQUE BENCHMARKS												
		Wood			Östegard		Fahle			ILOG Solver		
Name	$\|K\|$	$\|K\|$	#select	time	$\|K\|$	time	$\|K\|$	#select	time	$\|K\|$	#select	time
brock200_1	21	21	379,810	53.68	21	18.10	21	66,042	92.97	21	93,795	10.72
brock200_2	12	12	2,594	0.26	12	0	12	437	0.31	12	2,185	0.29
brock200_3	15	15	24,113	2.57	15	0.15	15	2,332	2.23	15	7,821	0.86
brock200_4	17	17	52,332	6.20	17	0.33	17	8,779	8.18	17	23,037	2.13
brock400_1	27		fail			fail	≥ 24	fail		27	60,159,630	11,340.8
brock400_2	29		fail			fail	≥ 29	fail		29	36,843,872	7,910.6
brock400_3	31		fail			fail	≥ 24	fail		31	19,616,188	4,477.23
brock400_4	33		fail			fail	≥ 25	fail		33	32,457,068	6,051.77
brock800_1	23		fail			fail	≥ 21	fail		≥ 21	fail	
brock800_2	24		fail			fail	≥ 20	fail		≥ 20	fail	
brock800_3	25		fail			fail	≥ 20	fail		≥ 20	fail	
brock800_4	26		fail			fail	≥ 20	fail		≥ 20	fail	
c-fat200-1	12	12	8	0	12	0	12	5	0	12	3	0
c-fat200-2	24	24	7	0	24	0	24	5	0	24	3	0
c-fat200-5	58	58	27	0	58	2.6	58	5	0	58	3	0
c-fat500-1	14	14	13	0	14	0.02	14	3	0	14	3	0
c-fat500-10	126	126	1	0	126	0.02	126	5	0.02	126	3	0.04
c-fat500-2	26	26	23	0	26	0.03	26	5	0	26	3	0
c-fat500-5	64	64	23	0	64	3,480.21	64	5	0.02	64	3	0
hamming10-2	512	512	1	0	512	0.84	512	257	5.16	512	257	1.04
hamming10-4	≥ 40		fail			fail	≥ 32	fail		≥ 40	fail	
hamming6-2	32	32	1	0	32	0	32	17	0	32	17	0
hamming6-4	4	4	81	0	4	0	4	31	0	4	42	0
hamming8-2	128	128	1	0	128	0	128	65	0.07	128	65	0
hamming8-4	16	16	36,441	5.28	16	0.28	16	1,950	6.11	16	40,078	4.19
johnson16-2-4	8	8	256,099	13.05	8	0.09	8	126,460	7.91	8	250,505	3.80
johnson32-2-4	≥ 16		fail			fail	≥ 16	fail		≥ 16	fail	
johnson8-2-4	4	4	23	0	4	0	4	15	0	4	14	0
johnson8-4-4	14	14	115	0	14	0	14	39	0.03	14	140	0
keller4	11	11	12,829	1.23	11	0.17	11	1,771	2.53	11	7,871	0.5
keller5	27		fail			fail	≥ 25	fail		≥ 27	fail	
keller6	≥ 59		fail			fail	≥ 43	fail		≥ 54	fail	
MANN_a9	16	16	60	0	16	0	16	31	0	16	50	0
MANN_a27	126	126	47,264	46.95		fail	126	39,351	10,348.87	126	1,258,768	18.48
MANN_a45	345		fail			fail	≥ 331	fail		≥ 345	fail	
MANN_a81	≥1100		fail			fail	≥ 996	fail		≥ 1100	fail	
p_hat300-1	8	8	1,310	0.10	8	0	8	254	0.07	8	364	0.11
p_hat300-2	25	25	2,801	0.67	25	0.33	25	1,121	3.01	25	1,695	0.59
p_hat300-3	36		fail			fail	36	171,086	856.67	36	102,053	40.71
p_hat500-1	9	9	9.772	0.91	9	0.1	9	690	0.6	9	8,731	2.30
p_hat500-2	36	36	59,393	17.81	36	142.93	36	32,413	203.93	36	41,259	32.69
p_hat500-3	50		fail			fail	≥ 48	fail		50	10,986,526	12,744.7
p_hat700-1	11	11	25,805	2.69	11	0.22	11	2,195	2.67	11	25,653	6.01
p_hat700-2	44		fail			fail	44	188,823	2,086.63	44	259,775	255.79
p_hat700-3	≥ 62		fail			fail	≥ 54	fail		≥ 62	fail	
p_hat1000-1	10	10	179,082	18.88	10	1.95	10	19,430	16.43	10	69,582	27.80
p_hat1000-2	46		fail			fail	≥ 44	fail		46	14,735,370	16,845.7
p_hat1000-3	≥ 68		fail			fail	≥ 50	fail		≥ 66	fail	
p_hat1500-1	12		fail			fail	12	136,620	119.77	12	1,063,765	480.84
p_hat1500-2	≥ 65		fail			fail	≥ 52	fail		≥ 63	fail	
p_hat1500-3	≥ 94		fail			fail	≥ 56	fail		≥ 91	fail	
san1000	15	15	106,823	43.59	15	0.17	15	35,189	3044.09	15	256,529	102.80
san200_0.7_1	30	30	206	0.06	30	0.19	30	301	1.57	30	1,310	0.36
san200_0.7_2	18	18	195	0.03	18	0	18	394	0.66	18	3,824	0.37
san200_0.9_1	70	70	2,069	0.77	70	0.09	70	20,239	62.61	70	1,040	1.04
san200_0.9_2	60	60	211,889	70.13	60	1.43	60	309,378	1930.90	60	6,638	2.62
san200_0.9_3	44		fail			fail	44	32,327	194.96	44	758,545	182.70
san400_0.5_1	13	13	3,465	0.75	13	0	13	882	6.74	13	6,204	1.19
san400_0.7_1	40	40	38,989	13.25		fail	40	11,830	425.99	40	70,601	23.28
san400_0.7_2	30	30	1,591,030	415.12	30	168.7	30	26,818	159.72	30	249,836	67.53
san400_0.7_3	22		fail			fail	22	213,195	617.07	22	1,690,023	273.23
san400_0.9_1	100		fail			fail	100	291,195	7,219.53	100	984,133	1,700
sanr200_0.7		18	150,861	22.50	18	4.7	18	25,582	24.99	18	41,773	4.30
sanr200_0.9			fail			fail	≥ 41	fail		42	541,496	150.08
sanr400_0.5		13	233,381	22.55	13	2.21	13	32,883	23.09	13	164,276	17.12
sanr400_0.7			fail			fail	21	9,759,158	15,925	21	22,791,798	3,139.11

Our program reaches the best lower bounds found so far for 6 problems: p_hat700-3, johnson32-2-4, hamming10-4, keller5 (the optimal value is reached), MANN_a45 (the optimal value is reached), and MANN_a81.

Better lower bounds have been found by [1] for p_hat1500-2 (65) , and p_hat1500-3 (94); and by [8] for keller6 (59) and p_hat1000-3 (68).

We think that this method is not the good one to solve some problems: keller6, johnson32-2-4, hamming10-4. For the other open problems, we are more confident.

5.1 Comparison with complete methods

We compare our approach with 3 other algorithms:
• [11]: A branch-and-bound approach using fractionnal coloring and lower bound heuristic.
• [9]: this approach is similar to dynamic programming: solve the problem with one node, then with 2 nodes, and so on until reaching n. Each time the optimal value of the previous computations is used as a minimal value for the new problem.
• [7]: This is the first CP approach. Fahle proposes to consider two filtering: the first one consists of removing the nodes that have a degree which is too small to improve the current objective value, the second consists of computing an upper bound of the clique involving each vertex taken separately by using a well known heuristic algorithm for graph coloring. The strategy selects the node with the smallest degree.

We decided to use a normalization of the time of the other approaches, instead of re-program the algorithm, because we were able to compare the performance of our algorithm on several machines and then to obtain a time ratio that should be fair. Therefore we have used the following time ratio:
• The times given by Wood are divided by 15
• The times given by Östegard are divided by 3
• The times given by Fahle are divided by 1.5

We can resume the comparison with other complete method by the following table:

	Wood	Östegard	Fahle	ILOG Solver
number of solved problems	38	36	45	52
number of problems solved in less than 10 min.	38	35	38	44
number of best time	15	26	10	30
number of best lower bound for open problems	0	0	1	5

If we consider all the problems solved by Östegard in less than 10 minutes, then Östegard needs 345.88s for solving all these problems, whereas we need only 282.44s

Our approach is almost always better than the Fahle's one, only p_hat1500-1 is quickly solved by the Fahle's method.

5.2 Comparison with heuristic methods

Two heuristic methods give very interesting results for solving the maximum clique problem: QUALEX-MS [5] and the method proposed in [10].

For the set of benchmarks we consider, these two methods are able to reach the best bound known so far for 50 problems. In less than 1 minute QUALEX-MS found 48 best bounds.

Here are the results we obtain with our approach:

– Within a limit of 4 hours of computation, our method is able to reach the best bound for 58 problems (and for 52 the optimality is proved).
– In less than 10 minutes of computation, we are able to find 49 best bounds, whose 44 are proved to be optimal.
– Within a limit of 1 minute of computation, we can reach 41 best bounds, and prove that 37 are optimal.

These results show that our method is competitive in regards to the best heuristic methods, even when the computational time is limited.

5.3 Interest of the diving technique

The diving technique is used after 10 minutes of computations. This technique requires most of the time less than 10 s, except for some huge problems where 100 s are needed and for keller6 which needs one and half hour.

It improves the current objective value $|K|$ found so far by the search, for 4 problems:
• brock400_2: the current value is 24, and the diving technique gives 29. From this information the search is speed-up. Without the diving technique we need 9,163 s to solve the problem, whereas with it, we need only 7,910 s.
• keller6: the current value is 51, and the diving technique gives 54. This result is interesting because after 4 hours of computation the solver is not able to improve 51. Therefore the diving technique in itself gives a better result. This result cannot be improved in 4 hours of computation.
• p_hat1500-3: the current value is 89, and the diving technique gives 91. This value cannot be improved by further computations within the limit of 4 hours.
• san400_0.9_1: the current value is 92. The diving technique gives 100 which is the optimal value. With the diving technique 1,700 s are needed to solve the problem, instead of 2,900 s.

6 A Maximum clique constraint

We can imagine to have a constraint stating the a set of nodes of a graph must be a clique of size greater than a given integer K. For instance, this set of nodes can be represented by a set variable as presented in ILOG Solver. Then, the filtering algorithm associated with this constraint will aim to remove some values of this set variable. The current set will then be defined by the required elements of the set variable. Moreover, then a node will be required all its non-neighboor will be removed from the possible set. In this case Algorithm 3 can be used as a filtering algorithm.

Moreover, a *Not* set could also be used. For instance, it could be given at the definition, and the growing of this set could be managed. We can define a filtering algorithm involving this set by using Property 9. The dominance properties cannot be used because some solutions could be missed.

7 Conclusion

In this paper we have presented a CP approach to solve a famous combinatorial optimization problem. We have presented new upper bound for the maximum clique problem and adapted and generalized the ideas of Bron and Kerbosh to this problem. The results that we obtain are good: seven problems are closed and 2 lower bounds have been improved for problems remaining open. We have also discussed the possible definition of a maximum clique constraint and its association with a filtering algorithm based on the ones presented in this paper. We hope that our ideas will lead to new improvements of the CP approach. In order, to encourage these improvements we claim that our approach is, currently, one of the best methods to solve the Maximum Clique Problem.

References

1. E. Balas and W. Niehaus. Finding large cliques in arbitrary graphs by bipartite matching. In D. Johnson and M. Trick, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 29–52. American Mathematical Society, 1996.
2. C. Berge. *Graphe et Hypergraphes*. Dunod, Paris, 1970.
3. I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. *Handbook of Combinatorial Optimization*, 4, 1999.
4. C. Bron and J. Kerbosh. Algorithm 457 : Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.
5. S. Busygin. A new trust region technique for the maximum weight clique problem. *Submitted to Special Issue of Discrete Applied Mathematics: Combinatorial Optimization*, 2002.
6. Dimacs. Dimacs clique benchmark instances. *ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique*, 1993.
7. Torsten Fahle. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In R. Möring and R. Raman, editors, *ESA 2002, 10th Annual European Symposium*, pages 485–498, 2002.
8. S. Homer and M. Peinado. Experiements with polynomial-time clique approximation algorithms on very large graphs. In D. Johnson and M. Trick, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 147–168. American Mathematical Society, 1996.
9. P Östegard. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, page to appear.
10. P. St-Louis, B. Gendron, and J. Ferland. A penalty-evaporation heuristic in a decomposition method for the maximum clique problem. In *Optimization Days*, Montreal, Canada, 2003.
11. D. Wood. An algorithm for finding maximum clique in a graph. *Operations Research Letters*, 21:211–217, 1997.