MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **58** (2007) 569-590

ISSN 0340 - 6253

An improved branch and bound algorithm for the maximum clique problem

Janez Konc and Dušanka Janežič*

National Institute of Chemistry, Hajdrihova 19, SI-1000 Ljubljana, Slovenia

(Received June 21, 2007)

Abstract

A new algorithm for finding a maximum clique in an undirected graph is described. An approximate coloring algorithm has been improved and used to provide bounds to the size of the maximum clique in a basic algorithm which finds a maximum clique. This basic algorithm was then extended to include dynamically varying bounds. The resulting algorithm is significantly faster than the comparable algorithm.

* Author to whom correspondence should be addressed
email: dusa@cmm.ki.si
telephone: 00386-14760200
fax: 00386-14760300

1 Introduction

A clique is a subset S of vertices in a graph G such that each pair of vertices in S is connected by an edge. The maximum clique problem is the problem of finding in a given graph the clique with the largest number of vertices. The algorithms for finding a maximum clique are frequently used in chemical information, bioinformatics and computational biology applications [1], where their main application is to search for similarity between molecules.

These algorithms are used for screening databases of compounds to filter out molecules that are similar to known biologically active molecules and are feasible to be active themselves [2]. Also, these algorithms are used for comparing protein structures, to provide the information about protein function [3] and also the information about possible interactions between proteins [4, 5].

Searching for the maximum clique is often the bottle-neck computational step in these applications. The maximum clique problem is NP-hard [6], and probably no polynomial time algorithm will be possible, but improvements to the existing algorithms can still be effective. Exact algorithms, which can be guaranteed to find the maximum clique, usually use a branch-and-bound approach to the maximum clique problem [7, 8, 9, 10], searching systematically through possible solutions and applying bounds to limit the search space. The tightest bounds come from the vertex-coloring method. This method assigns colors to vertices, so that no two adjacent vertices of a graph G are colored with the same color. The number of colors is the upper bound to the size of the maximum clique in graph G. Vertex-coloring is also known to be NP-hard [6], so a graph can only be colored approximately.

In this paper, we present improvements to an *approximate coloring algorithm* [8]. We use this algorithm in a basic algorithm for finding a maximum clique [7, 8], where the coloring algorithm provides upper bounds to the size of the maximum clique. The effects on the speed of the maximum clique algorithm of varying the tightness of upper bounds *dynamically* during the search are investigated experimentally. We make use of this concept of varying upper bounds to enhance the performance of this algorithm in agreement with our modified approximate coloring algorithm. The idea is that the tightest and the most computationally demanding upper bounds should be calculated close to the root of the recursion tree of the branch-and-bound algorithm, while on subsequent levels, where the majority of the search takes place, more relaxed and less computationally expensive bounds should be used. Our algorithm has been tested on random graphs and benchmark graphs, which were developed as part of the Second DIMACS Challenge [11], and is compared to the recent leading algorithm [8]. The improvements to the approximate coloring algorithm, together with the dynamical use of upper bounds, reduce the number of steps required to find the maximum clique and improve the run time of the algorithm by as much as an order of magnitude on dense graphs, while preserving its superior performance on sparse graphs.

2 Theory

2.1 Notations

An undirected graph $G = (V, E)$ consists of a set of vertices $V = \{1, 2, ..., n\}$ and a set of edges $E \subseteq V \times V$. Two vertices v and w are adjacent, if there exists an edge $(v, w) \in E$. For a vertex $v \in V$, a set $\Gamma(v)$ is the set of all vertices $w \in V$ that are adjacent to the vertex v. $|\Gamma(v)|$ is the degree of vertex v. The maximum degree in G is denoted as $\Delta(G)$. Let $G(R) = (R, E \cap R \times R)$ be the subgraph induced by vertices in R, where R is a subset of V. The density of a graph is calculated as $D = |E|/(|V| \cdot (|V| - 1)/2)$. The number of vertices in a maximum clique is denoted by $\omega(G)$.

2.2 The basic algorithm

A well known basic algorithm for finding a maximum clique [8] (*MaxClique*) is shown in Figure 1.

```
Procedure MaxClique($R, C$)
 1.    while $R \neq \emptyset$ do
 2.        choose a vertex $p$ with a maximum color $C(p)$ from set $R$;
 3.        $R := R \backslash \{p\}$;
 4.        if $|Q| + C(p) > |Q_{max}|$ then
 5.            $Q := Q \cup \{p\}$;
 6.            if $R \cap \Gamma(p) \neq \emptyset$ then
 7.                obtain a vertex-coloring $C'$ of $G(R \cap \Gamma(p))$;
 8.                MaxClique($R \cap \Gamma(p)$, $C'$);
 9.            else if $|Q| > |Q_{max}|$ then$Q_{max} := Q$;
10.            $Q := Q \backslash \{p\}$;
11.        else return
12.    end while
```

Figure 1. The basic maximum clique algorithm.

The algorithm *MaxClique* maintains two global sets Q and Q_{max}, where Q consists of vertices of the currently growing clique and Q_{max} consists of vertices of the largest clique currently found. The algorithm starts with an empty set Q, and then recursively adds vertices to (and deletes vertices from) this set, until it can verify that no clique with more vertices can be found. The next vertex to be added to Q is selected from the set of candidate vertices $R \subseteq V$, which is initially set to $R := V$. At each step, the algorithm selects a vertex $p \in R$ with the maximum color $C(p)$ among the vertices in R, and deletes it from R. $C(p)$ is the upper bound to the size of the maximum clique in the resulting set R. If the sum $|Q| + C(p)$ indicates that a clique larger than the one currently in Q_{max} can be found in R, then vertex p is added to the set Q. The new candidate set $R \cap \Gamma(p)$ with the corresponding vertex-coloring C' is calculated and then passed as parameters to the recursive call to the *MaxClique* procedure. If $R_p = \emptyset$ and $|Q| > |Q_{max}|$, *i.e.*, the current clique is larger than the currently largest clique found, then the vertices of

Q are copied to Q_{max}. The algorithm then backtracks by removing p from Q and then selects the next vertex from R. This procedure continues until $R = \emptyset$.

2.3 Approximate coloring algorithm

The *approximate coloring algorithm* introduced in Tomita and Seki (2003), provides vertex-coloring in the $MaxClique$ algorithm. All vertices are colored in the candidate set one by one in the order in which they appear in this set. The algorithm inserts each vertex $v \in R$ into the first possible color class C_k, so that v is non-adjacent to all the vertices already in this color class. If the current vertex v has at least one adjacent vertex in each color class $C_1, ..., C_k$, then a new color class C_{k+1} is opened and vertex v is inserted here. After all vertices in R have been assigned to their respective color classes, these vertices are then copied from the color classes as they appear in each color class C_k, and in the increasing order with respect to index k, back to R. In this process, a color $C(v) = k$ is assigned to each vertex $v \in R$. The outputs of the algorithm are the new set R and the vertex-coloring C, where colors in the set C correspond to vertices in R.

The number of color classes, which is the upper bound to the size of the maximum clique in the graph induced by R, depend heavily on how vertices are presented to this algorithm. The upper bound is tighter (lower), when vertices in R are considered in a non-increasing order with respect to their degree in G [8, 12].

2.4 Improved approximate coloring algorithm

We have improved the algorithm described above, which is referred to as the *original approximate coloring algorithm*, so as to maintain the non-increasing order of vertices in the candidate set R. This means that the order of vertices in R after the application of our coloring algorithm to this set is the same to the order of vertices in R prior to the algorithm start. This is not the case with the original approximate coloring algorithm, which orders the vertices in R by their colors, so that the $MaxClique$ algorithm can at each step select a vertex with a maximum color from the set R, which is conveniently the last vertex in this set. We observe that vertices $v \in R$ with colors $C(v) < |Q_{max}| - |Q| + 1$, need not be ordered by their colors as the $MaxClique$ algorithm will never add these vertices to the current clique Q (line 4, Figure 1). An inherent property of these vertices is that their colors are lower than a certain color, which we denote k_{min}. We introduce a counter j of such vertices. At the start of the approximate coloring algorithm we calculate color $k_{min} := |Q_{max}| - |Q| + 1$ and we set $j := 0$. If $k_{min} \leq 0$, then we set $k_{min} := 1$, because colors are positive numbers. When in the loop a vertex v at the $i - th$ position in R is assigned to a color class C_k, we test if $k < k_{min}$ for this vertex. If this is so, then we shift this vertex v from the $i - th$ to the $j - th$ position in R and we increase j by 1. When the assignment of vertices to color classes is complete, vertices with colors $k < k_{min}$ are at the front of the set R in their initial non-increasing order with respect to their degrees in G. The remaining vertices are copied from color classes C_k, where $k \geq k_{min}$, back to R as they appear in each color class C_k and in increasing order with respect to index k. Here, only these vertices are assigned colors $C(v) = k$. This algorithm **ColorSort** is shown in Figure 2.

2.5 Coloring example

Figure 3 depicts an undirected graph. The candidate set of vertices in the non-increasing order with respect to their degrees (in parentheses) is $R = \{7^{(5)}, 1^{(4)}, 4^{(4)}, 2^{(3)}, 3^{(3)}, 6^{(3)}, 5^{(2)}, 8^{(2)}\}$. This set is the input to the approximate coloring algorithm. In Table 1 vertices of the example graph are assigned to color classes; this procedure is the same for both approximate coloring algorithms.

```
Procedure ColorSort($R, C$)
 1.  $max\_no := 1$;
 2.  $k_{min} := |Q_{max}| - |Q| + 1$;
 3.  if $k_{min} \leq 0$ then $k_{min} := 1$;
 4.  $j := 0$;
 5.  $C_1 := \emptyset$; $C_2 := \emptyset$;
 6.  for $i := 0$ to $|R| - 1$ do
 7.      $p := R[i]$; {the $i$-th vertex in $R$}
 8.      $k := 1$;
 9.      while $C_k \cap \Gamma(p) \neq \emptyset$ do
10.          $k := k + 1$;
11.      if $k > maxno$ then
12.          $maxno := k$;
13.          $C_{maxno+1} := \emptyset$;
14.      end if
15.      $C_k := C_k \cup \{p\}$;
16.      if $k < k_{min}$ then
17.          $R[j] := R[i]$;
18.          $j := j + 1$;
19.      end if
20.  end for
21.  $C[j - 1] := 0$;
22.  for $k := k_{min}$ to $max\_no$ do
23.      for $i := 1$ to $|C_k|$ do
24.          $R[j] := C_k[i]$;
25.          $C[j] := k$;
26.          $j := j + 1$;
27.      end for
28.  end for
```

Figure 2. Improved approximate coloring algorithm.

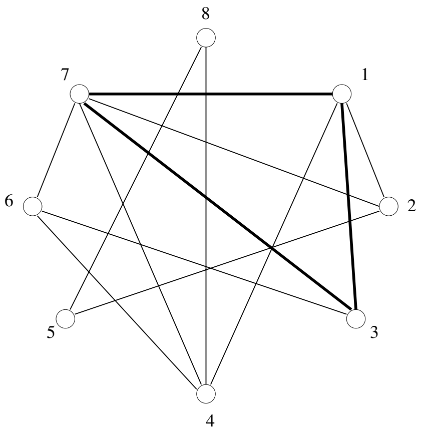

Figure 3. Example of an undirected graph. A maximum clique ($\omega = 3$) is depicted with its edges emphasized.

After assigning vertices to color classes, the *original coloring algorithm* copies vertices from these classes back to the candidate set, which becomes $R = \{7^{(5)}, 5^{(2)}, 1^{(4)}, 6^{(3)}, 8^{(2)}, 4^{(4)}, 2^{(3)}, 3^{(3)}\}$ with the respective coloring $C = \{1, 1, 1, 2, 2, 3, 3, 3\}$. Compared to the set R at the start of this coloring algorithm in this new set R vertices are less ordered with respect to their degrees. We expect that when the original coloring algorithm is used with the $MaxClique$ algorithm, the disorder of vertices in the resulting candidate sets will accumulate on the following levels of the recursion. The original approximate coloring algorithm no longer considers vertices which are sorted by their degrees, which is not efficient.

In the case of our $ColorSort$ algorithm, we set as an example $|Q_{max}| := 2$ and $|Q| := 0$. k_{min} is calculated by $k_{min} := 2 - 0 + 1 = 3$. In the loop where the assignment of vertices to color classes takes place, our algorithm shifts vertices with colors $k < 3$ to the front of the set R. After all the vertices have been assigned to color classes, the partial new candidate set is $R = \{7, 1, 6, 5, 8\}$. The remaining vertices are in color classes with $k \geq 3$, and in this case only vertices from the color class C_3, are

Table 1. Vertices of the example graph assigned to color classes with the approximate coloring algorithm. In each row are vertices of the color class C_k, where index $k \in \mathbb{N}$ is a color of these vertices. The degrees are in parentheses.

k	C_k
1	$7^{(5)}$ $5^{(2)}$
2	$1^{(4)}$ $6^{(3)}$ $8^{(2)}$
3	$4^{(4)}$ $2^{(3)}$ $3^{(3)}$

copied to R in the order in which they appear in this color class. The final candidate set is then $R = \{7^{(5)}, 1^{(4)}, 6^{(3)}, 5^{(2)}, 8^{(2)}, 4^{(4)}, 2^{(3)}, 3^{(3)}\}$ and the coloring $C = \{-, -, -, -, -, 3, 3, 3\}$, where a $-$ indicates that no color has been assigned to the corresponding vertex in set R. It can be seen that the vertices in this candidate set $\{7^{(5)}, 1^{(4)}, 6^{(3)}, 5^{(2)}, 8^{(2)}\}$ are now in decreasing order with respect to their degree, in contrast to the set R obtained with the original approximate coloring algorithm, where the same vertices $\{7^{(5)}, 5^{(2)}, 1^{(4)}, 6^{(3)}, 8^{(2)}\}$ are not ordered. In our computational experiments on various graphs, the number of vertices in the candidate sets with $k < k_{min}$ is on average much higher than number of vertices where the opposite is true. Therefore initial non-increasing order is maintained for most vertices in these candidate sets.

2.6 Dynamic coloring

Until now, in the $MaxClique$ algorithm the calculation of the degrees and sorting of vertices was performed only once with the initial set of vertices V. The coloring algorithms considered vertices in the candidate set R sorted by their degrees in G. An alternative to this is to recalculate at each and every *step* of the $MaxClique$ algorithm the degrees of vertices in R in the graph induced by these vertices, *i.e.*, $G(R)$, and sort these vertices in a non-increasing order with respect to their degrees in $G(R)$. Then the $ColorSort$ algorithm considers vertices in R sorted by their degrees in the induced graph

$G(R)$ rather than in G. The upper bounds given by this coloring algorithm are then as tight as possible with this approach. The number of steps required to find the maximum clique is reduced to the minimum, but the overall running time of the *MaxClique* algorithm does not improve, because of the computational expense $O(|R|^2)$ of the determination of the degrees and sorting of vertices in R.

We assume that improvement in the performance can be achieved by sorting vertices by their degrees in $G(R)$ only when the candidate set R is sufficiently large. Obviously, set R is larger on initial levels of the *MaxClique* algorithm. With the *level* of the *MaxClique* algorithm we denote the number of branches (recursive calls) from the root to the current leaf of the recursion tree. This is because, for large candidate sets the computational expense related to the computation of tighter bound is much smaller than the cost of investigating false solutions, which arise when applying less tight bounds. The same is not true for small candidate sets, where tighter bounds are much less effective in reducing the redundant searching and the computational expense related to the calculation of tighter upper bounds becomes significant.

The number of levels up to which the calculation of the degrees and sorting improves the speed of the maximum clique algorithm has to be determined dynamically, during the search for the maximum clique. For example, in dense graphs, maximum cliques are generally larger than in sparse graphs of equal size. We expect that the number of levels up to which tighter bounds should be used will be higher for dense than for sparse graphs. We also expect that for large graphs this number will be higher than for small graphs of equal density, as larger maximum cliques are generally found in large graphs. In Figure 4 is shown the **MaxCliqueDyn** algorithm, which we explain in details below.

We introduce global variables $S[level]$ and $S_{old}[level]$, which hold the sum of steps the *MaxCliqueDyn* algorithm performs from the root node up to

```
Procedure MaxCliqueDyn($R, C, level$)
 1.   $S[level] := S[level] + S[level-1] - S_{old}[level]$;
 2.   $S_{old}[level] := S[level-1]$;
 3.   while $R \neq \emptyset$ do
 4.       choose a vertex $p$ with maximum C(p) (last vertex) from $R$;
 5.       $R := R \setminus \{p\}$;
 6.       if $|Q| + C[index\ of\ p\ in\ R] > |Q_{max}|$ then
 7.           $Q := Q \cup \{p\}$;
 8.           if $R \cap \Gamma(p) \neq \emptyset$ then
 9.               if $S[level]/ALL\_STEPS < T_{limit}$ then
10.                   calculate the degrees of vertices in $G(R \cap \Gamma(p))$;
11.                   sort vertices in $R \cap \Gamma(p)$ in a descending order
12.                   with respect to their degrees;
13.               end if
14.               ColorSort($R \cap \Gamma(p), C'$)
15.               $S[level] := S[level] + 1$;
16.               $ALL\_STEPS := ALL\_STEPS + 1$;
17.               MaxCliqueDyn($R \cap \Gamma(p), C', level + 1$);
18.           else if $|Q| > |Q_{max}|$ then $Q_{max} := Q$;
19.           $Q := Q \setminus \{p\}$;
20.       else return
21.   end while
```

Figure 4. Maximum clique algorithm with dynamically varying upper bounds.

and including the current level and the sum of steps up to and including the previous level, respectively. We introduce $T[level]$, which is the fraction of steps up to the current *level* among all the steps completed so far. We recalculate $T[level] = S[level]/ALL\_STEPS$ on each and every step, where ALL_STEPS is a global counter of steps, which is increased by 1 at each step of the *MaxCliqueDyn* algorithm. With a new parameter, which we call T_{limit}, we can limit the use of tighter bounds to certain levels. While $T[level] < T_{limit}$, we perform the calculations of the degrees and sorting and in the *ColorSort* algorithm we consider vertices in R sorted by their degrees in $G(R)$. When $T[level] \geq T_{limit}$, no additional calculations are performed.

At each *level* the *MaxCliqueDyn* algorithm first updates the sum of steps up to this level by $S[level] := S[level] + S[level-1] - S_{old}[level]$ and the sum of steps up to the previous level by $S_{old}[level] := S[level-1]$. Each time the algorithm advances to the next recursive level, $S[level]$ is increased by 1. An example of this calculation is shown in Table 2.

Table 2. Counting the steps. Columns represent levels of the recursion. The path of the algorithm is represented by right arrows (recursive calls) and down left facing arrows (backtracks). The notation is $S[level]_{(S_{old}[level])}$. Number $S[3] = 5$ is calculated as $S[3] = S[3] + S[2] - S_{old}[3] = 2 + 4 - 2 = 4$ and, because a recursive call follows (right arrow), $S[3] = S[3] + 1 = 5$. $S_{old}[3] = S[2] = 4$.

$$\begin{tikzcd}
\text{level} & 1 & 2 & 3 & \dots \\
& 1_{(0)} \arrow[r] & 2_{(1)} \arrow[r] & 2_{(2)} \arrow[dl] & \\
& & 2_{(1)} \arrow[dl] & & \\
& 2_{(0)} \arrow[r] & 4_{(2)} \arrow[r] & 5_{(4)} \arrow[r] & {} \\
& & \vdots & &
\end{tikzcd}$$

2.7 Experimental determination of the T_{limit} parameter

We determined the parameter T_{limit} by experiments on random graphs. We construct a random graph with n vertices by inserting an edge with probability p between each pair of its vertices. We also construct 10 graphs for each size n and probability p, where n is in the interval $100 - 500$ and p is in the interval $0.2 - 0.99$. These random graphs were then the input to the $MaxCliqueDyn$ algorithm. For each graph, we let $MaxCliqueDyn$ algorithm find the maximum clique multiple times, each time with a different parameter T_{limit} in the interval 0.0 to 1.0. When this parameter was set to $T_{limit} = 0.0$, no tight bounds were used, as opposed to when T_{limit} was set to 1.0, when the tight bounds were used at every step. We plotted the time to find the maximum clique against T_{limit} for each n and p and the scaled plots for some of the random graphs and for some of the DIMACS graphs are shown in Figures 5 and 6, respectively.

All the curves for dense graphs in these plots exhibit a minimum when T_{limit} is close to 0.05. For sparse graphs the optimal T_{limit} is 0.0, but in almost all these cases the additional calculations make the algorithm less than 10% slower (see Table 3).

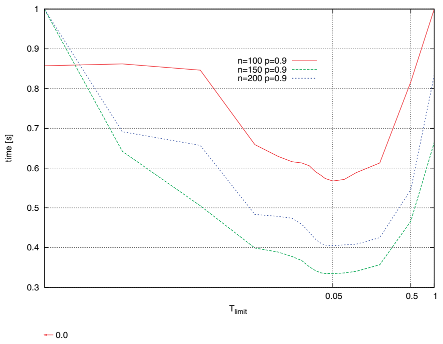

Figure 5. The graph showing the effect of varying T_{limit} on the time $MaxCliqueDyn$ algorithm requires to find a maximum clique in random graphs with $100, 150$, and 200 vertices with $p = 0.9$. A logarithmic scale is used on the x - axis.

We choose $T_{limit} = 0.025$ as higher values of this parameter increase the time of the calculation for sparse graphs and a lower T_{limit} makes the algorithm slower on very dense graphs. Other values of T_{limit} could be chosen in an interval (see Table 3) with little change in the time needed to find a maximum clique with the $MaxCliqueDyn$ algorithm. The choice of T_{limit} depends on the way the degrees and sorting of vertices are calculated. We calculate the degrees from ground up and use very simple $O(|R|^2)$ implementations of the sorting function. With this parameter set to $T_{limit} = 0.025$, the computationally expensive calculations of degrees and sorting are performed on about 2.5% of the steps of the $MaxCliqueDyn$ algorithm, although the true number of such steps may vary, because $T[level]$ is calculated dynamically during the search. Adapting to the local nature of the graph under consideration $T[level]$ can raise above or fall below T_{limit} in the search process, thereby

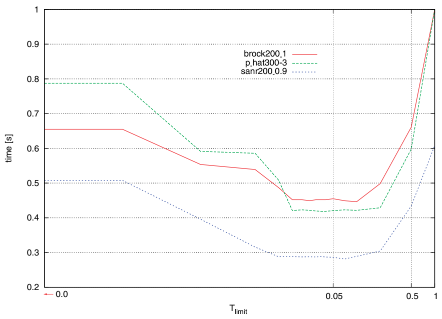

Figure 6. The graph showing the effect of varying T_{limit} on the time $MaxCliqueDyn$ algorithm requires to find a maximum clique in some of the DIMACS graphs. A logarithmic scale is used on the x - axis.

disallowing or permitting the calculation of tighter bounds.

Table 3. Intervals of values of the parameter T, where the run-time is within 10% of the minimum time. The sizes n are in the upper row and the probabilities p of graphs are in the left-most column. Where a clear minimum was observed, it is indicated in boldface.

$p\backslash n$	100	150	200	300	500
0.2	0-1	0-0.2	0-0.2	0-0.1	0-0.03
0.3	0-0.5	0-0.2	0-0.5	0-0.03	0-0.005
0.4	0-0.2	0-0.05	0-0.035	0-0.015	0-**0.02**-0.1
0.5	0-0.05	0-0.03	0-**0.01**-0.2	0.001-**0.035**-0.1	0.0001-**0.02**-0.2
0.6	0-0.05	0-0.1	0.005-**0.03**-0.07	0.001-**0.02**-0.2	0.005-**0.035**-0.07
0.7	0.01-**0.035**-0.2	0.005-**0.025**-0.1	0.015-**0.02**-0.2	0.005-**0.05**-0.2	--
0.8	0.005-**0.03**-0.1	0.01-**0.03**-0.1	0.005-**0.05**-0.2	0.005-**0.05**-0.2	--
0.9	0.015-**0.05**-0.2	0.02-**0.05**-0.2	0.025-**0.05**-0.2	--	--
0.95	0.01-**0.035**-0.07	0.01-**0.03**-0.1	0.015-**0.05**-0.1	--	--
0.99	0-1	0-1	0-1	0-1	0-1

2.8 Initialization

We set $Q := \emptyset$, $Q_{max} := \emptyset$ and $ALL\_STEPS := 1$. The elements of sets S and S_{old} are set to 0. We calculate the degrees of vertices V in graph G and sort these vertices in a non-increasing order with respect to these degrees.

The first $\Delta(G)$ vertices in V are colored with numbers $1 \ldots \Delta(G)$ and the rest of the vertices in V are assigned a color $\Delta(G)$. The first candidate set V, together with its coloring C, is then an input to the $MaxCliqueDyn$ procedure. This initialization follows the standard procedure described elsewhere [8].

3 Experimental results

Our $MaxCliqueDyn$ algorithm was implemented in C. Computational experiments were performed on a 1.6GHz AMD Opteron processor with Linux operating system. We also implemented Tomita's maximum clique finding algorithm MCQ [8]. First, we compared the basic maximum clique algorithm $MaxClique$ with our improved approximate coloring algorithm $ColorSort$ to the MCQ algorithm. Then we compared our $MaxCliqueDyn$ algorithm to the MCQ algorithm on random graphs, and finally, we performed the same comparison on DIMACS graphs. Our user times for the DIMACS machine benchmark graphs r100.5-r500.5 are 0.00, 0.04, 0.40, 2.48, and 9.45 seconds, respectively. We set the limit for the time available to the algorithms to 21600 seconds ($= 6h$).

Our improved approximate coloring algorithm $ColorSort$ reduces the number of steps to find the maximum clique on random graphs when compared to the original approximate coloring algorithm, when both algorithm are used to provide upper bounds in the basic algorithm $MaxClique$. The reduction is most apparent for difficult graphs with p in the interval $0.7 - 0.95$, *e.g.*, for random graphs with 200 vertices and $p = 0.95$ the number of steps is reduced 2.1 times. For graphs with higher or lower density the number of steps is reduced less. However these graphs in our random test set can be solved in very few steps and because no additional calculation is associated with the modifications to the original approximate coloring algorithm, the computing time decreases proportionately with the numbers of steps. The times and numbers of steps for the $MaxClique$ algorithm using $ColorSort$

algorithm are shown in Table 4.

We have determined the effect of our dynamic calculation of upper bounds on the performance of the *MaxCliqueDyn* algorithm, compared to the *MCQ* algorithm on random graphs. Table 4 shows average CPU times for random graphs obtained by both algorithms.

The data in Table 4 suggest that our *MaxCliqueDyn* algorithm is $1.3 - 12$ times faster than the *MCQ* algorithm on random graphs with probabilities in the interval $0.7 - 0.95$; the speed-up tends to increase for larger graphs. With other graphs, the difference between the algorithms is smaller. Our algorithm is slightly slower on certain sparse graphs, where a small number of unnecessery calculations of degrees and sorting of vertices is performed. This is in part because $T[level]$ does not adapt to the local properties of these graphs.

In Table 5 we show the results of *MaxCliqueDyn* and *MCQ* algorithms on DIMACS graphs. Because the *MCQ* algorithm was originally tested only on some of these graphs [8], we tested this algorithm again. The calculation times of our algorithm confirm that the $T_{limit} = 0.025$ is well chosen for these graphs, as the calculation times are close to the minimum for most of these graphs. We solve all four very difficult `brock800` family instances in under 4 hours compared to the *MCQ* algorithm, which solves only 2 of these instances and requires considerably more time. We also solved the `p_hat700-3` graph in under 4 hours; in this case the *MCQ* algorithm fails to complete. Times are improved for all but one graph in the brock family, where our times are generally better by a factor $2-3$. With our algorithm, `p_hat1000-2` is solved 6 times faster, `p_hat300-3` 3 times, `p_hat500-3` almost 8 times, `san1000` 7 times, `san200_0.9_1` almost 6 times, `san400_0.7_2` 6 times, `san400_0.9_1` 31 times and `sanr200_0.9` is solved almost 6 times faster.

Our maximum clique algorithm does not improve times for very dense ($d \geq 0.99$) graphs in the `MANN` family and for `hamming10-2  graph.`Table 5 shows that in these cases numbers of steps are unchanged. A small number of

Table 4. CPU times $[s]$ and numbers of steps for random graphs; p is the probability of an edge between two vertices of a graph with n vertices.

Graph		MCQ		MaxClique+CS		MaxCliqueDyn+CS	
n	p	num. steps	CPU time	num. steps	CPU time	num. steps	CPU time
100	0.6	1031	0.003	978	0.00265	943	0.0028
100	0.7	2752	0.00925	2525	0.0082	1940	0.00735
100	0.8	6674	0.03	5460	0.02445	4101	0.0205
100	0.9	9279	0.0746	6655	0.05405	4314	0.0384
100	0.95	764	0.0092	712	0.00815	477	0.0061
150	0.5	2526	0.0076	2423	0.00695	2217	0.00735
150	0.6	7852	0.0278	7330	0.02525	5932	0.0238
150	0.7	30057	0.1315	27087	0.1164	20475	0.09445
150	0.8	218972	1.2913	174905	1.0368	88649	0.606
150	0.9	1570126	17.478	854110	9.863	258853	3.464
150	0.95	78246	1.7241	40081	0.88255	14825	0.40785
200	0.4	2585	0.00795	2494	0.00745	2409	0.0078
200	0.5	9774	0.0329	9355	0.0304	7695	0.03095
200	0.6	40694	0.1626	38060	0.14845	31114	0.12995
200	0.7	275430	1.383	246297	1.22485	148726	0.8599
200	0.8	3555022	25.725	2850467	20.658	1424940	11.06
200	0.9	145553091	2117.78	80498743	1199.39	16404111	279.705
200	0.95	22393742	729.479	10433846	340.077	1508657	60.524
300	0.4	15014	0.0507	14591	0.0479	12368	0.05335
300	0.5	76112	0.30745	73311	0.28725	61960	0.2634
300	0.6	641131	2.983	603540	2.767	387959	2.133
300	0.7	8521492	51.915	7669432	46.385	4548515	28.566
300	0.8	530043775	4580.5	434614352	3735	159498760	1442.01
500	0.3	23300	0.09055	22785	0.0863	19665	0.10895
500	0.4	141095	0.6202	137348	0.58595	123175	0.5817
500	0.5	1422379	7.009	1368896	6.579	963385	5.827
500	0.6	23889293	142.553	22678735	132.084	15075757	91.447
1000	0.2	45171	0.29045	44766	0.2753	41531	0.3858
1000	0.3	456758	2.386	449717	2.238	413895	2.332
1000	0.4	6192174	34.708	6040135	32.914	4332149	33.761
1000	0.5	140261760	886.465	136018698	842.86	100756405	655.6

Table 5. CPU times $[s]$ and numbers of steps for DIMACS benchmark graphs; ω is the size of the largest clique found.

Graph	MCQ			MaxCliqueDyn		
name	ω	num. steps	CPU time	ω	num. steps	CPU time
brock200_1	21	450327	2.74	21	229597	1.56
brock200_2	12	4232	0.017	12	3566	0.018
brock200_3	15	17089	0.088	15	13057	0.0735
brock200_4	17	64332	0.313	17	48329	0.242
brock400_1	27	326153861	2915	27	125736892	1175.68
brock400_2	29	115680020	1204.86	29	44010239	521.01
brock400_3	31	279192244	2297	31	109522985	935.23
brock400_4	33	129575982	1181.96	33	53669377	532.66
brock800_1	≥ 23	956318168	fail	23	1445025793	14344
brock800_2	≥ 24	1071802831	fail	24	1304457116	13507
brock800_3	25	1581256139	17050	25	835391899	9263
brock800_4	26	1105720024	13142	26	564323367	7180
c-fat200-1	12	214	0.0005	12	214	0.0005
c-fat200-2	24	239	0.001	24	239	0.0005
c-fat200-5	58	307	0.003	58	307	0.003
c-fat500-10	126	743	0.0355	126	743	0.0345
c-fat500-1	14	517	0.0015	14	517	0.002
c-fat500-2	26	542	0.0025	26	542	0.003
c-fat500-5	64	618	0.0095	64	618	0.0095
hamming6-2	32	62	0.0005	32	62	0.0005
hamming6-4	4	105	0.0005	4	105	0
hamming8-2	128	254	0.018	128	254	0.018
hamming8-4	16	41603	0.333	16	19107	0.1505
hamming10-2	512	1022	1.3	512	2048	6.63
hamming10-4	≥ 40	16274629	fail	≥ 40	1934328	fail
johnson8-2-4	4	46	0	4	46	0
johnson8-4-4	14	255	0.001	14	221	0.001
johnson16-2-4	8	430130	0.4365	8	643573	0.681
johnson32-2-4	≥ 16	15	fail	≥ 16	15	fail
keller4	11	12209	0.0485	11	8991	0.04
keller5	≥ 27	162625	fail	≥ 27	25921	fail
keller6	≥ 50	40912647	fail	≥ 52	319878688	fail
MANN_a9	16	94	0.0005	16	94	0.0005
MANN_a27	126	38252	6.92	126	38252	7.56
MANN_a45	345	2852231	5480	345	2852231	9037

Graph	MCQ			MaxCliqueDyn		
name	ω	num. steps	CPU time	ω	num. steps	CPU time
p_hat300-1	8	2137	0.006	8	2084	0.0055
p_hat300-2	25	9944	0.0795	25	7611	0.063
p_hat300-3	36	2519267	30.62	36	629972	9.1
p_hat500-1	9	11275	0.043	9	10933	0.041
p_hat500-2	36	515253	6.57	36	189060	2.81
p_hat500-3	50	239705791	5683	50	25599649	739.16
p_hat700-1	11	34229	0.17	11	27931	0.19
p_hat700-2	44	4355991	87.49	44	1071470	25.4
p_hat700-3	≥ 57	486128224	fail	62	292408292	13583
p_hat1000-1	10	202628	1.04	10	170203	0.9855
p_hat1000-2	46	218545180	5189	46	30842192	859.44
p_hat1000-3	≥ 53	798989871	fail	≥ 59	481023263	fail
p_hat1500-1	12	1283326	8.39	12	1138496	7.75
p_hat1500-2	≥ 55	494607640	fail	≥ 60	405328276	fail
p_hat1500-3	≥ 60	239190830	fail	≥ 69	389162339	fail
san200_0.7_1	30	1542	0.019	30	983	0.0125
san200_0.7_2	18	1577	0.011	18	1750	0.014
san200_0.9_1	70	268746	2.86	70	28678	0.4815
san200_0.9_2	60	489203	5.52	60	72041	1.48
san200_0.9_3	44	1037194	17.9	44	327704	6.66
san400_0.5_1	13	4182	0.0525	13	3026	0.0255
san400_0.7_1	40	135677	2.84	40	47332	0.8195
san400_0.7_2	30	71937	1.8	30	9805	0.286
san400_0.7_3	22	411539	5.8	22	366505	3.21
san400_0.9_1	100	41072828	2108	100	697695	66.43
san1000	15	224566	11.37	15	114537	1.56
sanr200_0.7	18	182244	0.924	18	104996	0.586
sanr200_0.9	42	41107366	583.26	42	6394315	102.66
sanr400_0.5	13	299625	1.34	13	248369	1.14
sanr400_0.7	21	89775740	602.9	21	37806745	287.78

calculations of the degrees and sorting are still performed, and this accounts for the increased calculation time.

Notably, a discrepancy between times of our version of the *MCQ* algorithm and times obtained by authors exist for random graphs. It seems that they limited the maximum clique size within their random graphs. Although we used a different computer, our times are directly comparable for the DIMACS benchmark graphs, except for the `san400_0.9_1` graph, in which the initial order of vertices has a particular effect on the performance of the maximum clique algorithms we tested. This may be because the last vertex, *i.e.*, the

400th vertex, is a member of the maximum clique. With minor modifications to our sorting algorithm, the MCQ algorithm required 722 seconds and our $MaxCliqueDyn$ algorithm 53.44 seconds.

4 Conclusions

In this paper we describe an improved approximate coloring algorithm and an algorithm for finding a maximum clique in an undirected graph. We show that by applying tighter, more computationally expensive upper bounds on a fraction of the search space, it is possible to reduce the time to find the maximum clique. Our algorithm, which has not been fine-tuned, is considerably faster than the original MCQ algorithm. A similar strategy is feasible for other types of upper bounds [9], where advantage could be taken of a trade-off between expensive computation and overall speed.

Acknowledgement

The financial support through grants P1-0002 of the Ministry of Higher Education, Science, and Technology of Slovenia is acknowledged. The authors thank Prof. G.W.A. Milne for critical reading of the manuscript.

References

[1] S. Butenko, W.E. Wilhelm, Clique-detection models in computational biochemistry and genomics, European Journal of Operational Research 173 (2006) 1-17.

[2] C. Hofbauer, H. Lohninger, A. Aszódi, SURFCOMP: A novel graph-based approach to molecular surface comparison, Journal of Chemical Information and Computer Science 44 (2004) 837-847.

[3] S. Schmitt, D. Kuhn, G. Klebe, A new method to detect related function among proteins independent of sequence and fold homology, Journal of Molecular Biology 323 (2002) 387-406.

[4] J. Konc, D. Janežič, A branch and bound algorithm for matching protein structures, Lecture Notes in Computer Science 4432 (2007) 399-406.

[5] J. Konc, D. Janežič, Protein-protein binding-sites prediction by protein surface structure conservation, Journal of Chemical Information and Modeling 47 (2007) 940-944.

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

[7] D.R. Wood, An algorithm for finding a maximum clique in a Graph, Operations Research Letters 21 (1997) 211-217.

[8] E. Tomita, T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique, Lecture Notes in Computer Science 2631 (2003) 278-289.

[9] E. Balas, J. Xue, Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring, Algorithmica 15 (1996) 397-412.

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

[11] D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability: Second DIMACS Challenge, DIMACS Series in Disrete Mathematics and Theoretical Computer Science, vol. 26. American Mathematical Society, Providence, 1996.

[12] N. Biggs, Some heuristics for graph colouring, in: R. Nelson, R.J. Wilson (Eds.), Graph colourings, Longman, New York 1990, pp. 87-96.