J Glob Optim (2007) 37:95–111
DOI 10.1007/s10898-006-9039-7

ORIGINAL PAPER

An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments

Etsuji Tomita · Toshikatsu Kameda

Received: 12 September 2005 / Accepted: 30 April 2006 /
Published online: 6 July 2006

Abstract We present an exact and efficient branch-and-bound algorithm MCR for finding a maximum clique in an arbitrary graph. The algorithm is not specialized for any particular type of graph. It employs approximate coloring to obtain an upper bound on the size of a maximum clique along with an improved appropriate sorting of vertices. We demonstrate by computational experiments on random graphs with up to 15,000 vertices and on DIMACS benchmark graphs that in general, our algorithm decidedly outperforms other existing algorithms. The algorithm has been successfully applied to interesting problems in bioinformatics, image processing, design of quantum circuits, and design of DNA and RNA sequences for biomolecular computation.

Keywords Maximum clique · Algorithm · Branch-and-bound · Approximate coloring · Computational experiments

1 Introduction

Given an undirected graph G, a *clique* is a subgraph of G, in which all the pairs of vertices are adjacent. Finding a maximum clique in a graph is one of the most important NP-hard problems, and it has been studied by many researchers. Pardalos and Xue (1994) and Bomze et al. (1999) presented excellent surveys of this problem. See also *Chap. 7: Selected Applications* in Bomze et al. (1999) for applications of maximum clique algorithms.

Some remarkable developments have been made by Tarjan and Trojanowsky (1977), Robson (1986), and others to improve the *theoretical order* of the time-complexity to solve the maximum clique problem. These algorithms, however, are not

E. Tomita (✉) · T. Kameda
Department of Information and Communication Engineering, The University of Electro-Communications, Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, Japan
e-mail: tomita@ice.uec.ac.jp

T. Kameda
e-mail: kam@ice.uec.ac.jp

necessarily fast in solving problems of practical size. Since, this problem has important practical applications, another approach with practically fast performance is required. A collection of studies in this direction was presented in Johnson and Trick (1996). Our present paper also addresses the issue of speed.

One of the standard approaches for finding a maximum clique is based on the branch-and-bound method. Several branch-and-bound algorithms use approximate coloring to obtain an upper bound on the size of a maximum clique. Elaborate coloring can significantly reduce the search space. However, coloring is time-consuming, and therefore, it becomes important to choose an appropriate trade-off between the time required for approximate coloring and the reduction in the search space thereby obtained. Many efforts have been made along this line (see Bomze et al. 1999). Recently, Östergård (2002) proposed a new maximum clique algorithm, which was supported by computational experiments. Sewell (1998) presented a maximum clique algorithm designed for dense graphs.

In this paper, we present a branch-and-bound algorithm MCR for finding a maximum clique. This algorithm is based on approximate coloring and appropriate sorting of the vertices. We experimentally compare MCR with the other existing algorithms. The experimental results show that our algorithm is very fast for many random graphs and DIMACS benchmark graphs. The tested graphs include large random graphs with up to 15,000 vertices.

The algorithm MCR is an improved algorithm of MCQ that was presented in Tomita and Seki (2003). These algorithms yielded interesting results for the applications in bioinformatics by Bahadur et al. (2002, 2005, 2006) and Akutsu et al. (2006), and in other areas by Hotta et al. (2003), Nakui et al. (2003), and Kobayashi et al. (2003).

2 Preliminaries

1. Throughout this paper, we are concerned with a simple undirected graph $G = (V, E)$ with a finite set V of vertices and a finite set E of *unordered* pairs $(v, w)(= (w, v))$ of distinct vertices called edges. The set V of vertices is considered to be *ordered*, and the ith element in V is denoted by $V[i]$. A pair of vertices v and w are said to be adjacent if $(v, w) \in E$.
2. For a subset $W \subseteq V$ of vertices, $G(W) = (W, E(W))$ with $E(W) = \{(v, w) \in W \times W | (v, w) \in E\}$ is called a subgraph of $G = (V, E)$ *induced* by W.
3. For a vertex $v \in V$ and $V' \subseteq V$, let $\Gamma_{V'}(v)$ be the set of all vertices that are adjacent to v in an induced subgraph $G(V')$, i.e.,

$$\Gamma_{V'}(v) = \{w \in V' | (v, w) \in E\} \ (\not\ni v).$$

   We call $deg_{V'}(v) = |\Gamma_{V'}(v)|$, the number of vertices adjacent to a vertex v in $G(V')$, the degree of v in $G(V')$. If $V' = V$ or V' is understood from the context, then $\Gamma_{V'}(v)$ and $deg_{V'}(v)$ may be simply written as $\Gamma(v)$ and $deg(v)$, respectively. In general, the number of elements in a set S is denoted by $|S|$.
4. Given a subset $Q \subseteq V$ of vertices, the induced subgraph $G(Q)$ is said to be a clique if $(v, w) \in E$ for all $v, w \in Q$ with $v \neq w$. In this case, we may simply say that Q is a clique. In particular, a clique of the maximum size is called a *maximum clique*. The number of vertices of a maximum clique in graph $G = (V, E)$ is denoted by

$\omega(G)$ or $\omega(V)$. Note that $\omega(G) \leq \Delta(G) + 1$, where $\Delta(G)$ is the maximum degree in graph G.

A subset $W \subseteq V$ of vertices is said to be independent if $(v, w) \notin E$ for all $v, w \in W$.

3 Maximum clique algorithm MCR

3.1 A basic algorithm

Our basic algorithm begins with a small clique, and continues finding larger and larger cliques until one is found that can be verified to have the maximum size. More precisely, we maintain global variables Q and $Q_{\max}$, where Q consists of vertices of a current clique, and $Q_{\max}$ consists of vertices of the largest clique found so far. Let $R \subseteq V$ consist of vertices (candidates) that may be added to Q. We begin the algorithm by letting $Q := \emptyset$, $Q_{\max} := \emptyset$, and $R := V$ (the set of all vertices). We select a certain vertex p from R and add p to Q ($Q := Q \cup \{p\}$). Then, we compute $R_p := R \cap \Gamma(p)$ as the new set of candidate vertices. This procedure (EXPAND in Fig. 1, where each statement with a comment /∗ . . . ∗/ at its right is to be replaced by the statement (possibly empty) between /∗ and ∗/ in this basic stage.) is applied recursively while $R_{\mathrm{p}} \neq \emptyset$.

When $R_{\mathrm{p}} = \emptyset$ is reached, Q constitutes a maximal clique. If Q is maximal and $|Q| > |Q_{\max}|$ holds, $Q_{\max}$ is replaced by Q. We then backtrack by removing p from Q and R. We select a new vertex p from the resulting R and continue the same procedure

```
procedure EXPAND(R, No)                          /* procedure EXPAND(R) */
begin
    while R ≠ ∅ do
        p := the vertex in R
                such that No[p] = Max{No[q] | q ∈ R};  /*                  */
                {i.e., the last (rightmost) vertex in R}  /*                */
        if |Q| + No[p] > |Q_max|                 /* if |Q| + |R| > |Q_max| */
            then                                 /*                          */
            Q := Q ∪ {p};
            R_p := R ∩ Γ(p);
            if R_p ≠ ∅ then
                NUMBER-SORT(R_p, No');           /*                          */
                {the initial value of No' has no significance}  /*          */
                EXPAND(R_p, No')                 /*   EXPAND(R_p)            */
            else if |Q| > |Q_max| then Q_max := Q fi
            fi
            Q := Q − {p}
        else return
        fi
        R := R − {p}
    od
end {of EXPAND}
```

Fig. 1 EXPAND (/∗ Basic EXPAND ∗/)

until $R = \emptyset$. This is a well-known basic algorithm for finding a maximum clique (for example, Fujii and Tomita 1982; Tomita et al. 1988; Carrghan and Pardalos 1990; Tomita et al. to appear, and others).

3.2 Pruning

In order to prune unnecessary searching, we employ the *approximate coloring* of vertices, as introduced in Tomita and Yamada (1978), Fujii and Tomita (1982), and Tomita et al. (1988). We assign in advance for each $p \in R$ a positive integer value $No[p]$ called the *Number* or *Color* of p with the following property:

(1) If $(p, r) \in E$ then $No[p] \neq No[r]$, and
(2) $No[p] = 1$, or if $No[p] = k > 1$, then there exist vertices $p_1 \in \Gamma(p), p_2 \in \Gamma(p), \ldots, p_{k-1} \in \Gamma(p)$ in R with $No[p_1] = 1, No[p_2] = 2, \ldots, No[p_{k-1}] = k - 1$.

Consequently, we know that

$$\omega(R) \leq \mathrm{Max}\{No[p] | p \in R\}$$

and hence, if $|Q| + \mathrm{Max}\{No[p] | p \in R\} \leq |Q_{\max}|$ holds then, we can disregard such R.

```
procedure NUMBER-SORT(R, No)
begin
{NUMBER}
    maxno := 0;
    C_1 := ∅;
    while R ≠ ∅ do
        p := the first vertex in R;
        k := 1;
        while C_k ∩ Γ(p) ≠ ∅
            do k := k + 1 od
        if k > maxno then
            maxno := k;
            C_maxno := ∅
        fi
        No[p] := k;
        C_k := C_k ∪ {p};
        R := R − {p}
    od
{SORT}
    i := 1;
    for k := 1 to maxno do
        for j := 1 to |C_k| do
            R[i] := C_k[j];
            i := i + 1
        od
    od
end {of NUMBER-SORT}
```

Fig. 2 Number-sort

The value $No[p]$ for every $p \in R$ can be easily assigned step by step by a so-called *greedy coloring* algorithm as follows: Assume the vertices in $R = \{p_1, p_2, \ldots, p_m\}$ are arranged in this order. First, let $No[p_1] = 1$. Next, let $No[p_2] = 2$ if $p_2 \in \Gamma(p_1)$, else $No[p_1] = 1, \ldots$, and so on. After *Numbers* are assigned to all vertices in R, we sort these vertices in ascending order with respect to their *Numbers*. We call this numbering and sorting procedure NUMBER-SORT (see Fig. 2 for details of Procedure NUMBER-SORT). This procedure runs in $O(|R|^2)$ time. Note that the quality of such *sequential* coloring depends heavily on how the vertices are ordered. Therefore, the SORT portion of NUMBER-SORT is important. Hence, we consider a simple and effective sorting procedure as follows.

Let $\mathrm{Max}\{No[r] | r \in R\} = maxno$,

$$C_i = \{r \in R | No[r] = i\}, \quad i = 1, 2, \ldots, \mathrm{maxno}$$

and

$$R = C_1 \cup C_2 \cup \cdots \cup C_{\mathrm{maxno}}$$

where the vertices in R are *ordered* in a manner such that first appear the vertices in C_1, in the same order as that in C_1, and then, the vertices in C_2 follow in a similar manner, and so on.

Let

$$C'_i(p) = C_i \cap \Gamma(p), \quad i = 1, 2, \ldots, \mathrm{maxno},$$

for some $p \in R$, and we have

$$R_p = R \cap \Gamma(p) = C'_1(p) \cup C'_2(p) \cup \cdots \cup C'_{\mathrm{maxno}}(p),$$

where the vertices in R_p are ordered in the manner described above. Both C_i and $C'_i(p)$ are independent sets, and $C'_i(p) \subseteq C_i$ for $i = 1, 2, \ldots, \mathrm{maxno}$. Thus, it is evident that the maximum *Number* (*Color*) required for newly coloring $C'_i(p)$ is less than or equal to that required for C_i. This indicates that the above coloring is steadily improved step by step owing to the procedure NUMBER-SORT. In addition, it is noteworthy that the latter part {SORT} in Fig. 2 runs in *only* $O(|R|)$ time.

A more elaborate coloring may be more effective in reducing the total search space; however our preliminary computational experiments indicate that elaborate coloring schemes require a considerable amount of computation time, and thus they have an overall negative effect on the performance.

In procedure EXPAND(R, No), after applying NUMBER-SORT more than once, a maximum clique contains a vertex p in R such that $No[p] \geq \omega(R)$. It is generally expected that a vertex p in R such that $No[p] = \mathrm{Max}\{No[q] | q \in R\}$ has a high probability of belonging to a maximum clique. Accordingly, we select a vertex p in R such that $No[p] = \mathrm{Max}\{No[q] | q \in R\}$, as described at the beginning of the **while** loop in EXPAND(R, No). Here, a vertex p such that $No[p] = \mathrm{Max}\{No[q] | q \in R\}$ is the *last* element in the ordered set R of vertices after the application of NUMBER-SORT. Therefore, we simply select the *rightmost* vertex p in R in $O(1)$ time while $R \neq \emptyset$. Consequently, the vertices in R are selected from the last (right) to the first (left).

3.3 Initial sorting and initial numbering

Fujii and Tomita (1982) demonstrated that both the search space and the overall running time are reduced when the vertices are sorted in an ascending order with respect

to their degrees prior to the application of the branch-and-bound algorithm for finding a maximum clique. Carrghan and Pardalos (1990) also employed a similar technique successfully. Therefore, at the beginning of our algorithm, we sort the vertices in V in a descending order with respect to their degrees. This indicates that a vertex p at the beginning of the **while** loop in EXPAND(V, No) is selected in an ascending order with respect to its degrees in V, since the selection is *from right to left*.

Let $G = (V, E)$ be a graph with $|V| = n$ and $V = \{V[1], V[2], \ldots, V[n]\}$, when vertices $V[n], V[n-1], \ldots, V[i+1]$ have been selected and removed from V successively, we are to consider a subgraph of G induced by $\{V[1], V[2], \ldots, V[i]\}$. Then, the size of a maximum clique of such an induced subgraph is less than or equal to $\mathrm{Min}\{i, \Delta(G) + 1\}$. Therefore, we initially assign *Numbers* to the vertices in V so that $No[V[i]] = i$ for $i \leq \Delta(G)$, and $No[V[i]] = \Delta(G) + 1$ for $\Delta(G) + 1 \leq i \leq |V|$. This initial *Number* has the desired property that in EXPAND(V, No), $No[p] \geq \omega(V)$ for any p in V **while** $V \neq \emptyset$, as stated above. Thus, this simple initial *Number* is sufficient. Figure 3 along with Figs. 1 and 2, completes the MCQ algorithm in Tomita and Seki (2003), where all comments beginning with /∗ and ending with ∗/ should be deleted from Fig. 1.

Note that the rather time-consuming calculation of the degree of vertices is carried out only at the beginning of MCQ and not in NUMBER-SORT. Therefore, the total time required to reduce the search space can be *very small*. It should be also noted that the initial order of the vertices in our algorithm is effective for the reduction in the search space, as described at the beginning of this section. Moreover, this effectiveness is also observed in the subsequent subproblems. This is because the initial order of the vertices in the same *Number* is *inherited* in the subsequent subproblems due to the method employed in NUMBER-SORT.

In order to confirm the effectiveness of our initial sorting, we have implemented an algorithm revMCQ, which for comparison is obtained from MCQ by replacing "Sort vertices of V in the *descending* order" in MCQ with "Sort vertices of V in the *ascending* (i.e., reverse) order." The results of some computations carried out by MCQ and revMCQ for random graphs with the number of vertices n and the edge probability p in Table 1 confirm that our initial sorting is effective. Here, Branches indicates the total

```
procedure MCQ (G = (V, E))
begin
    global Q := ∅;
    global Q_max := ∅;
{SORT}
    Sort vertices of V in a descending order with respect to their degrees;
{NUMBER}
    for i := 1 to Δ(G)
        do No[V[i]] := i od
    for i := Δ(G) + 1 to |V|
        do No[V[i]] := Δ(G) + 1 od
    EXPAND(V, No);
    output Q_max
end {of MCQ }
```

Fig. 3 Algorithm MCQ

Table 1 MCQ versus revMCQ

Graph			CPU time [sec]		Branches	
n	p	ω	MCQ	revMCQ	MCQ	revMCQ
100	0.9	29–32	0.07	3.60	10,854	582,642
300	0.7	19–21	37.12	89.91	6,214,907	13,495,981
600	0.5	14	17.81	24.35	3,427,350	4,222,273
2000	0.3	10–11	63.14	71.69	9,261,982	9,736,627

number of EXPAND() calls excluding that at the beginning. Hence, it corresponds to the extent of the search space. Each entry under CPU time and Branches is an average value for 10 graphs with each pair of n and p. Entries in column ω are the range of the size of the maximum cliques obtained. The computer used for the experiment is identical to that used in Sect. 4.

3.4 Algorithm MCR with improved initial sorting

We further improve the *initial* sorting of the vertices. First, we alter the order of the vertices in $V = \{V[1], V[2], \ldots, V[n]\}$ such that in a subgraph of $G = (V, E)$ induced by a set of vertices $V' = \{V[1], V[2], \ldots, V[i]\}$, it holds that $V[i]$ always has the minimum degree in $\{V[1], V[2], \ldots, V[i]\}$ for $1 \leq i \leq |V|$. While this order, as a result, is similar to that of Carraghan and Pardalos (1990), it should be noted again that time-consuming calculation of the degree of vertices is carried out *only at the beginning* of MCR as in MCQ, and hence the overhead of the overall selection of vertices is *very small*, too.

In general, we have several vertices in V' with the same degree. In such a case, for a vertex q, we further consider adjacent vertices and define a value $ex\text{-}deg_{V'}(q) = \sum_{r \in \Gamma_{V'}(q)} deg_{V'}(r)$. Then, we arrange vertices $V[i-1]$ and $V[i]$ with the same degree so that $ex\text{-}deg_{V'}(V[i-1]) \geq ex\text{-}deg_{V'}(V[i])$.

Finally, when all the vertices $V[1], V[2], \ldots, V[i]$ have the same (minimum) degree, that is, a subgraph induced by $\{V[1], V[2], \ldots, V[i]\}$ is *regular*, sorting these vertices becomes pointless. In this case, we apply NUMBER-SORT to this set of vertices, and we make use of the resulting *Number*(*No*). If the maximum of the resulting numbers for $V[1], V[2], \ldots, V[i]$ is maxno, then we assign the vertices $V[i+1], V[i+2], \ldots, V[n]$, the initial *Numbers* Min $\{\text{maxno} + 1, \Delta(G) + 1\}$, Min $\{\text{maxno} + 2, \Delta(G) + 1\}$, …, Min $\{\text{maxno} + (|V| - i), \Delta(G) + 1\}$, respectively.

Note that if the vertices $V[1], V[2], \ldots, V[i]$ have the same degree $(i-1)$, then $V[1]$, $V[2], \ldots, V[i]$ constitutes a *clique* of size i.

By taking the above all into account, we have an improved clique finding algorithm MCR, as shown in Fig. 3.5.

See Sect. 3.5. **Example** in Tomita and Kameda (2005) for an example run of MCR. (See also *Example* in Sect. 3 of Tomita et al. (to appear) for understanding a *Basic Algorithm* in Sect. 3.1.)

3.5 Comparison between MCQ and MCR

Prior to the computational experiments for MCR, we present some characteristic results of computational experiments for the comparison between MCQ and MCR

```
procedure MCR(G = (V, E))
begin
    global Q := ∅; global Q_max := ∅;
{SORT}
    i := |V|;
    R := V; V := ∅;
    R_min := set of vertices with the minimum degree in R;
    while |R_min| ≠ |R| do
        if |R_min| ≥ 2 then
            p := a vertex in R_min such that ex-deg(p) = Min{ex-deg(q) | q ∈ R_min}
        else p := R_min[1];
        fi
        V[i] := p; R := R − {p};
        i := i − 1;
        for j := 1 to |R| do
            if R[j] is adjacent to p then
                deg(R[j]) := deg(R[j]) − 1
            fi
        od
        R_min := set of vertices with the minimum degree in R
    od
{Regular subgraph}
    NUMBER-SORT(R_min, No);
    for i := 1 to |R_min| do
        V[i] := R_min[i]
    od
{NUMBER}
    m := Max{No[q] | q ∈ R_min} ;
    mmax := |R_min| + Δ(G) − m;
    m := m + 1;
    i := |R_min| + 1;
    while i ≤ mmax do
        if i > |V| then goto Start fi
        No[V[i]] := m;
        m := m + 1;
        i := i + 1
    od
    for i := mmax + 1 to |V| do
        No[V[i]] := Δ(G) + 1
    od
Start:
    if deg_{R_min}(q) = |R_min| − 1 for all q ∈ R_min
        then Q_max := R_min
    fi
    EXPAND(V, No);
    output Q_max
end {of MCR}
```

Fig. 4 Algorithm MCR

Table 2 MCQ versus MCR for DIMACS benchmark graphs

Graph				CPU time [sec]	
Name	n	density	ω	MCQ	MCR
hamming8-2	256	0.969	128	0.021	0.0036
hamming10-2	1,024	0.990	512	1.82	0.33
johnson16-2-4	120	0.765	8	0.34	0.21
p_hat300-3	300	0.744	36	26.5	15.8
p_hat500-3	500	0.752	50	4,470	2,663
san200_0.9_3	200	0.900	44	16.41	0.24
san400_0.7_3	400	0.700	22	9.5	4.5
san400_0.9_1	400	0.900	100	72.7	5.3

in Table 2. The computer used for the experiments is identical to that used in Sect. 4. Table 2 presents the results for DIMACS benchmark graphs given in Johnson and Trick (1996). In this table, density represents the edge density of the graph, i.e. (the number of edges of the graph)/$\{n(n-1)/2\}$. The MCR is confirmed to be faster than MCQ in almost all the cases, while MCQ could be faster than MCR when the given graph is very sparse or very dense. Henceforth, we exclusively focus on MCR in the following sections.

4 Computational experiments

We have implemented the algorithm MCR in the programming language C and carried out computational experiments to evaluate it. The computer used has a Pentium4 2.20 GHz CPU and a Linux operating system. We compare the CPU time required by MCR with those by the other existing algorithms for the same problems. The comparison is made based on the well-established way in the Second DIMACS Implementation Challenge for Cliques, Coloring, and Satisfiability shown in Johnson and Trick (1996).

First, we obtained our user time T_1 [sec] required to solve each of the given five benchmark instances `r100.5`, `r200.5`, `r300.5`, `r400.5`, and `r500.5` on our computer by the benchmark program dfmax given by Applegate and Johnson (see Johnson and Trick 1996) as shown under T_1 of Table 5 in Appendix.

For the other algorithm New in Östergård (2002) for comparison, we took his user time T_2 [sec] out of Sect. 3. Experimental results (p. 203) of Östergård (2002) required to solve each of the same five benchmark instances on his computer by the same benchmark program dfmax as shown under T_2 of Table 5 in Appendix. From these values, we calculated the average of the ratio T_2/T_1 to be 2.85. That is, we consider that our computer is 2.85 times faster than Östergård (2002)'s computer. See the Appendix for the details along with the other algorithms for comparison.

4.1 Results for random graphs

Prior to the present work, it was confirmed in Tomita et al. (1988) that an earlier version of MCQ was faster than Balas and Yu (1986)'s algorithm by computational experiments for random graphs.

For each pair of n (the number of vertices) up to 15,000 and p (edge probability) in Table 3, random graphs are generated so that there exists an edge for each pair of vertices with probability p. Then the average CPU times [seconds] required to solve these graphs by dfmax and MCR are listed in Table 3. The averages are taken for ten random graphs for each pair of n and p. The exceptions are the star (*)-marked CPU time and the CPU times for $n \geq 5,000$, where each CPU time is for a single graph with the given n and p values. In addition, the CPU times by New in Östergård (2002) and COCR (COC, for short) in Sewell (1998) are added in Table 3; each CPU times is adjusted according to the ratio given in the Appendix. The bold-faced entry indicates the fastest time in a given row.

In this table, it is evident that MCR is the fastest for random graphs except for the cases where $[n = 150, p = 0.9]$, $[n = 200, p = 0.8]$ and $[n = 200, p = 0.9]$. In these cases, COCR is the fastest. It is specially designed for solving the maximum clique problem in *dense* graphs.

For reference, the associated branches by dfmax and MCR are listed in Table 5 of Tomita and Kameda (2005). As one of the characteristic examples, for random graphs with $n = 150$ and $p = 0.98$, the number of branches by MCR is 28,401 (an average for ten graphs), while those by dfmax is more than $4.29 \times 10^9 = 2^{32}$ (for a single graph). As the result, the CPU time by MCR is 0.36 seconds and that by dfmax is more than 10^5 seconds, as shown in Table 3. The algorithms dfmax and MCR are similar in principle, therefore, the difference demonstrates the effectiveness of the tighter bounding condition and the appropriate sorting of vertices adopted in MCR.

These results show that MCR is successful in general for obtaining a good trade-off between the increase in computation time and the reduction in the search space associated with approximate coloring and the appropriate sorting of vertices.

Thus far, it is widely recognized that dfmax is the fastest maximum clique algorithm for sparse graphs, as stated in Östergård (2002) and Fahle (2002). Table 3, however, shows that MCR is faster than dfmax for all the graphs tested, including very sparse graphs. It is to be noted that MCR is considerably faster than dfmax when the number of vertices is very large, even for sparse graphs.

4.2 Results for DIMACS benchmark graphs

Table 4 lists the CPU times required by dfmax and MCR to solve the DIMACS benchmark graphs given in Johnson and Trick (1996). In addition, the CPU times by New in Östergård (2002), χ+DF in Fahle (2002), MIPO (MIP, for short) in Balas et al. (1996), and SQUEEZE (SQU, for short) in Bourjolly et al. (1996) are added to Table 4; each of these CPU times is adjusted according to the ratio given in the Appendix. Target/3 (Ta/3, for short) is also added in the last column for reference; each time in this column is that of Target in Stix (2003) divided by three due to the reason described in the Appendix. The bold-faced entry indicates the fastest time among those obtained within the time limits in a given row. The results in Table 4 show that MCR is faster than dfmax except for only very "easy" graphs, which are solved in less than 0.007 seconds.

MCR is also faster than New for most graphs except for a few. For a more detailed comparison between MCR and New, we note that MCR is more than ten times faster than New for ten graphs, while New is more than 10 times faster than MCR for only two graphs in Table 4. In addition, MCR is more than or equal to 100 times faster than

Table 3 CPU time [sec] for random graphs

Graph			dfmax	MCR	New	COCR(COC)
n	*p*	ω				
	0.5	9–10	0.0019	**0.0010**		0.13
	0.6	11–13	0.0061	**0.0022**	0.0035	0.13
	0.7	14–16	0.0286	**0.0063**	0.011	0.18
100	0.8	19–21	0.22	**0.019**	0.10	0.24
	0.9	29–32	5.97	**0.059**	1.04	0.31
	0.95	39–46	40.94	**0.019**	0.31	
	0.98	56–61	37.55	**0.0015**		
	0.7	16–18	0.57	**0.099**		0.53
150	0.8	23	11.23	**0.77**		1.18
	0.9	36–39	1,743.70	7.61		**1.83**
	0.95	50–57	61,118.8*	**4.27**		
	0.98	75–84	$> 10^5$*	**0.36**		
	0.4	9–10	0.012	**0.006**	0.011	
	0.5	11–12	0.058	**0.022**	0.03	0.40
	0.6	14	0.46	**0.12**	0.27	0.82
200	0.7	18–19	6.18	**0.93**	4.75	2.59
	0.8	24–27	314.92	17.63	231.54	**13.66**
	0.9	40–44	$> 10^5$*	980.36		**57.84**
	0.4	9–10	0.078	**0.036**	0.074	
	0.5	12–13	0.59	**0.196**	0.32	1.78
300	0.6	15–16	7.83	**1.85**	5.50	7.83
	0.7	19–21	233.69	**30.89**	179.71	
	0.8	28–29	48,280.6*	**1,790.01**		
	0.2	7	0.018	**0.014**	0.03	
	0.3	8–9	0.13	**0.064**	0.13	
	0.4	11	1.02	**0.44**	0.94	
500	0.5	13–14	14.45	**4.52**	11.40	27.41
	0.6	17	399.22	**79.94**	288.10	
	0.7	22–23	37,796.9*	**3,810.1***		
	0.2	7–8	0.24	**0.18**	0.33	
	0.3	9–10	3.09	**1.57**	2.58	
1,000	0.4	12	51.92	**19.49**	36.46	
	0.5	15	1,766.85	**482.52**		
	0.6	20	160,028.3*	**27,095.7***		
	0.1	6–7	1.82	**1.37**		
	0.2	9	35.07	**26.05**		
3,000	0.3	11–12	1,116.54	**598.58**		
	0.4	14	71,256.7*	**27,244.1***		
	0.1	7	13.51	**11.87**		
5,000	0.2	9	531.65	**420.74**		
	0.3	12	28,315.34	**18,524.67**		
10,000	0.1	7	256.43	**185.88**		
	0.2	10	23,315.34	**16,874.75**		
15,000	0.1	8	1,354.64	**875.96**		

Table 4 CPU time [sec] for DIMACS benchmark graphs

Graphs		dfmax	MCR	New	χ+DF	COC	MIP	SQU	Ta/3
Name (n, density)	ω								
brock200_1 (200, 0.745)	21	24	● **2.4**	19	112			1,471	116
brock200_2 (200, 0.496)	12	0.05	**0.015**	0.018	0.37	0.54	847	60	0.3
brock200_3 (200, 0.605)	15	0.35	○ **0.071**	0.15	2.7			195	1.7
brock200_4 (200, 0.658)	17	1.5	**0.29**	0.34	9.9	1.4		419	6.0
brock400_1 (400, 0.75)	27	41,615	● **2,410**		>17,000				
brock400_2 (400, 0.75)	29	25,595	**986**		>17,000	>652			
brock400_3 (400, 0.75)	31	27,895	★ **1,625**		>17,000				
brock400_4 (400, 0.75)	33	20,063	**868**		>17,000	>652			
brock800_1 (800, 0.65)	23	$> 10^5$	**22,711**		>17,000				
brcok800_2 (800, 0.65)	24	$> 10^5$	**20,519**		>17,000	>652			
brock800_3 (800, 0.65)	25	$> 10^5$	**13,877**		>17,000				
brock800_4 (800, 0.65)	26	$> 10^5$	**9,669**		>17,000	>652			
c-fat200-1 (200, 0.077)	12	○ **0.0002**	0.00075	0.0035	0.008		22	3.2	
c-fat200-2 (200, 0.163)	24	○ **0.0003**	0.0014	0.0035	0.008		11	2.8	
c-fat200-5 (200, 0.426)	58	444	○ **0.0025**	2.7	0.008		61	1.7	
c-fat500-1 (500, 0.036)	14	○ **0.0010**	0.0042	0.025	0.016			51	
c-fat500-2 (500, 0.073)	26	● **0.0011**	0.0063	0.028	0.016			91	
c-fat500-5 (500, 0.186)	64	2.7	**0.017**	3,664	0.024			50	
c-fat500-10 (500, 0.374)	126	$> 10^5$	0.034	0.025	**0.024**			36	
hamming6-2 (64, 0.905)	32	0.0175	★ **0.00011**	0.0035	0.008		0.004		
hamming6-4 (64, 0.349)	4	0.00015	**0.00010**	0.0035			0.48	0.09	
hamming8-2 (256, 0.969)	128	$> 10^5$	○ **0.0036**	0.014	0.088		0.05		
hamming8-4 (256, 0.639)	16	3.1	**0.28**	0.30	7.37	1.6	45	1,290	
hamming10-2 (1,024, 0.990)	512	$> 10^5$	○ **0.33**	0.88	6.2			0.96	
johnson8-2-4 (28, 0.556)	4	0.00005	○ **0.00002**	0.0035			0.0003		
johnson8-4-4 (70, 0.768)	14	0.0071	● **0.00055**	0.0035	0.032		0.0045	0.35	

Table 4 continued

Graphs		dfmax	MCR	New	χ+DF	COC	MIP	SQU	Ta/3
Name (n, density)	ω								
johnson16-2-4 (120, 0.765)	8	1.2	0.21	0.095	9.6		★ **0.003**	777	
keller4 (171, 0.649)	11	0.62	○ **0.037**	0.18	3.1	0.67	184	237	
MANN_a9 (45, 0.927)	16	0.062	★ **0.00018**	0.0035	0.008				
MANN_a27 (378, 0.990)	126	$> 10^5$	**4.2**	>3,500	12,489	4.3		738	
MANN_a45 (1035, 0.996)	345	$> 10^5$	**6,213**	>3,500	>17,000				
p_hat300-1 (300, 0.244)	8	0.08	○ **0.0060**	0.014	0.088	0.69	1,429	82	
p_hat300-2 (300, 0.489)	25	1.0	● **0.043**	0.35	3.6	0.96		175	
p_hat300-3 (300, 0.744)	36	1,259	15.8		1,034	**8.5**		3,972	
p_hat500-1 (500, 0.253)	9	0.09	○ **0.038**	0.10	0.72			837	0.7
p_hat500-2 (500, 0.505)	36	219	★ **4.5**	151	246			2,582	>7,200
p_hat500-3 (500, 0.752)	50	$> 10^5$	○ **2,663**		>17,000				>7,200
p_hat700-1 (700, 0.249)	11	0.32	**0.14**	0.24	3.2	4.3		>7,513	2.3
p_hat700-2 (700, 0.498)	44	8,651	64		2,518	**40**			>7,200
p_hat1000-1 (1000, 0.245)	10	1.7	○ **0.75**	2.1	20				
p_hat1000-2 (1000, 0.490)	46	$> 10^5$	○ **3,512**		>17,000				
p_hat1500-1 (1500, 0.253)	12	16	○ **6.3**		145				
san200_0.7_1 (200, 0.700)	30	4,181	● **0.027**	0.20	1.9		0.33	311	0.3
san200_0.7_2 (200, 0.700)	18	26,879	**0.0095**	0.014	0.80		8.5	>7,513	2.0
san200_0.9_1 (200, 0.900)	70	$> 10^5$	1.8	0.095	76		**0.08**	0.43	7.7
san200_0.9_2 (200, 0.900)	60	$> 10^5$	6.2	1.5	2,330		● **0.24**	13	109
san200_0.9_3 (200, 0.900)	44	71,005	★★ **0.24**		235		24	431	>7,200
san400_0.5_1 (400, 0.500)	13	719	0.029	○ **0.011**	8.1		133		1.3
san400_0.7_1 (400, 0.700)	40	$> 10^5$	★ **2.3**	>3,500	514			40.7	
san400_0.7_2 (400, 0.700)	30	$> 10^5$	★★ **0.45**	178	193		787		189
san400_0.7_3 (400, 0.700)	22	$> 10^5$	★★ **4.5**		745				>7,200

Table 4 continued

Graphs		dfmax	MCR	New	χ+DF	COC	MIP	SQU	Ta/3
Name (n, density)	ω								
san400_0.9_1 (400, 0.900)	100	$> 10^5$	★★ **5.3**		8,712			3,240	>7,200
san1000 (1000, 0.502)	15	$> 10^5$	7.1	★ **0.18**	3,673				
sanr200_0.7 (200, 0.702)	18	5.0	● **0.79**	4.9	30			318	
sanr200_0.9 (200, 0.898)	42	$> 10^5$	★ **434**		>17,000				
sanr400_0.5 (400, 0.501)	13	3.5	○ **1.1**	2.3	28				
sanr400_0.7 (400, 0.700)	21	3,939	● **503**		19,218				
DSJC500.5 (500, 0.502)	13	18	○ **5.2**			29		>7,513	
DSJC1000.5 (1000, 0.500)	15	1,987	**486**			>652			

Entries indicated by ★★, ★, ●, and ○ represent those that are more than or equal to 100, 10, 5, and 2 times faster than all the others in the same row, respectively.

New for six graphs, while New is more than 100 times faster than or equal to MCR for no graph in Table 4.

Further, MCR is faster than χ+DF for all graphs except c-fat500-10 in Table 4.

MCR is faster than COCR for all graphs except p_hat300-3 and p_hat700-2 in Table 4.

MCR is faster than MIPO for all graphs except johnson16-2-4, san200_0.9_1, and san200_0.9_2 in Table 4.

MCR is faster than SQUEEZE for all graphs except san200_0.9_1 in Table 4.

MCR is faster than Target/3 for all instances in Table 4.

As a whole in Table 4, MCR is more than 10 times faster than all the other algorithms for ten graphs. In particular, MCR is more than or equal to 100 times faster than all the other algorithms for four graphs. On the other hand, New is more than ten times faster than all the other algorithms for only one graph, and MIPO is more than ten times faster than all the other algorithms for only 1 graph, and other algorithms are more than ten times faster than all the other algorithms for no graph.

From the results described in Sects. 4.1 and 4.2, we can summarize that in general, MCR decidedly outperforms the other existing algorithms cited here.

5 Concluding remarks

We have shown that our pruning technique by NUMBER-SORT based on greedy coloring is very effective, and hence, in general MCR decidedly outperforms other existing algorithms. By using more elaborate coloring, we can increase the performance for dense graphs but with possible deterioration for sparse graphs as in Sewell (1998). The high performance of MCR results from its *simplicity*. Particularly the simplicity of NUMBER-SORT along with the appropriately improved initial sorting and

the simple *Numbering* of vertices considerably enhances the performance of MCR. It is noticed that a simpler algorithm of Corno et al. (1995) could be faster than MCR for *very large sparse* graphs. (See, Tomita et al. 2006).

Our algorithms have already been successfully applied to solve some interesting problems in bioinformatics by Bahadur et al. (2002, 2005, 2006) and Akutsu et al. (2006), image processing by Hotta et al. (2003), design of quantum circuits by Nakui et al. (2003), design of DNA and RNA sequences for biomolecular computation by Kobayashi et al. (2003). Our techniques can also be applied for generating maximal cliques, as shown in Tomita et al. (to appear).

Acknowledgements We would like to express our gratitude to T. Fujii, Y.Kohata, and T. Seki for their contributions in early stages of this study. We thank T. Nakagawa, S. Urabe, Y. Sutani, and T. Higashi for their contributions in the computational experiments. We also acknowledge the useful discussions and comments by T. Akutsu, J. Tarui, T. Nishino, S. Kobayashi, and the referees. Many helpful detailed comments by E. Harley are especially appreciated. We wish to thank D. S. Johnson and M. Trick for their efforts in organizing the Second DIMACS Implementation Challenge for Cliques, Coloring, and Satisfiability (Johnson and Trick 1996). They made it easier for us to compare the results of different algorithms carried out on various computers. This research was partially supported by Grants-in-Aid for Scientific Research Nos. 13680435 and 16300001 from the Ministry of Education, Culture, Sports, Science and Technology, Japan and the Research Fund of the University of Electro-Communications to the Advanced Algorithms Research Laboratory where the authors belong to. The research was also provided a grant by the Funai Foundation for Information Technology.

Appendix

Clique Benchmark Results
Type of Machine: Pentium4, 2.20 GHz
Compiler and flags used: gcc -O2
Machine Benchmarks: Tables 5 and 6
For Östergård (2002)'s user time for in instances (T_2) and Sewell (1998)'s user time for instances (T_3), by excluding the values of T_2/T_1 and T_3/T_1 for `r100.5` and `r200.5`,

Table 5 Each user time for instances [sec]

Graph	MCR	New		COCR(COC)	
	Our	Östergård's		Sewell's	
	T_1	T_2	T_2/T_1	T_3	T_3/T_1
`r100.5`	2.13×10^{-3}	0.01	4.69	0.14	65.73
`r200.5`	6.35×10^{-2}	0.23	3.62	3.64	57.32
`r300.5`	0.562	1.52	2.70	31.10	55.34
`r400.5`	3.48	10.05	2.89	191.98	55.06
`r500.5`	13.3	39.41	2.96	734.99	55.11
	MCR	MIPO(MIP)		SQUEEZE(SQU)	
	Our	Balas et al.'s		Bourjolly et al.'s	
	T_1	T_4	T_4/T_1	T_5	T_5/T_1
`r100.5`	2.13×10^{-3}	0.09	42.25	0.02	9.39
`r200.5`	6.35×10^{-2}	2.16	34.03	0.75	11.81
`r300.5`	0.562	18.5	32.92	6.36	11.32
`r400.5`	3.48	114	32.76	39.60	11.38
`r500.5`	13.3	432	32.48	153.57	11.55

Table 6 Fahle's user time for DIMACS benchmark graphs [sec]

Graph	***Our*** T_1	***Fahle***'s T_7	T_7/T_1
brock200_1	23.96	28.82	1.203
brock200_4	1.47	1.73	1.177
hamming8-4	3.07	4.05	1.319
hamming16-2-4	1.21	1.51	1.248
p_hat300-2	1.04	1.25	1.202
p_hat1000-1	1.66	2.18	1.313
p_hat1500-1	15.57	20.77	1.334
san400_0.5_1	719.16	924.5	1.286
sanr200_0.7	5.02	5.96	1.187
sanr400_0.5	3.50	4.07	1.163

since these instances are extremely small, the average values of T_2/T_1 and T_3/T_1 are obtained as 2.85 and 55.2, respectively. For Balas et al. (1996)'s user time for instances (T_4) and Bourjolly et al. (1996)'s user time for instances (T_5), the average values of T_4/T_1 and T_5/T_1 are obtained as 33.0 and 11.5, respectively, by similarly excluding the values T_4/T_1 and T_5/T_1 for r100.5 (see Table 5).

χ+DF

Fahle (2002)'s dfmax time (T_7) for some DIMACS benchmark graphs vs. our dfmax time (T_1) are shown in Table 6 in Appendix, while Fahle (2002)'s dfmax time for any of the above five benchmark instances r100.5,..., r500.5 is not described in Fahle (2002). By considering these values, we obtain the average value of T_7/T_1 as 1.24.

Target

No result is described in Stix (2003) on the CPU time by the benchmark program dfmax. Stix (2003) uses a Pentium III 450MHz PC, while Fahle (2002) uses a PentiumIII 933 PC. If we consider that Stix's PC is 450/933 times as fast as Fahle's PC, then Stix's PC is $1/(1.24 \times (933/450)) = 1/2.57$ times as fast as ours. Therefore, for Stix's user time for instances (T_8), we consider that T_8/T_1 is approximately at most 3.

References

Akutsu, T., Hayashida, M., Bahadur, D.K.C., Tomita, E., Suzuki, J., Horimoto, K.: Dynamic programming and clique based approaches for protein threading with profiles and constraints, IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences **E89-A** 1215–1222 (2006): The preliminary version was presented in: Akutsu, T., Hayashida, M., Tomita, E., Suzuki, J., Horimoto, K.: Protein threading with profiles and constraints, Proc. IEEE Symposium on Bioinformatics and Bioengineering (BIBE 2004). pp. 537–544 (2004)

Bahadur, D.K.C., Akutsu, T., Tomita, E., Seki, T., Fujiyama, A.: Point matching under non-uniform distortions and protein side chain packing based on efficient maximum clique algorithms. Genome Inform. **13**, 143–152 (2002)

Bahadur, D.K.C., Tomita, E., Suzuki, J., Horimoto, K., Akutsu, T.: Protein side-chain packing problem: a maximum edge-weight clique algorithmic approach, J. Bioinform Comput. Biol. **3**, 103–126 (2005)

Bahadur, D.K.C., Tomita, E., Suzuki, J., Horimoto, K., Akutsu, T.: Protein threading with profiles and distance constraints using clique based algorithms. J. Bioinform. Comput. Biol. **4**, 19–42 (2006)

Balas, E., Yu, C.S.: Finding a maximum clique in an arbitrary graph. SIAM J. Comput. **15**, 1054–1068 (1986)
Balas, E., Ceria, S., Cornuéjols, G., Pataki, G.: Polyhedral methods for the maximum clique problem, In: Johnson and Trick (eds.) pp. 11–28 (1996)
Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo M.: The Maximum Clique Problem, In: Du, D.-Z., Pardalos, P.M. (eds.), Handbook of Combinatorial Optimization, Supplement Vol. A, pp. 1–74. Kluwer Academic Publishers, Dordrecht (1999)
Bourjolly, J.-M., Gill, P., Laporte, G., Mercure, H.: An exact quadratic 0-1 algorithm for the stable set problem, In: Johnson and Trick (eds.) pp. 53–73 (1996)
Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem. Oper. Res. Lett. **9**, 375–382 (1990)
Corno, F., Prinetto, P., Sonza Reorda, M.: Using symbolic techniques to find the maximum clique in very large sparse graphs. Proc. European Design and Test Conference(EDTC 1995). pp. 320–324 (1995)
Fahle, T.: Simple and fast: Improving a branch-and-bound algorithm for maximum clique. European Symposium on Algorithms 2002, LNCS 2461. pp. 485–498 (2002)
Fujii, T., Tomita, E.: On efficient algorithms for finding a maximum clique. Technical Report of IECE, AL81-113. pp. 25–34 (1982)
Hotta, K., Tomita, E., Takahashi, H.: A view-invariant human face detection method based on maximum cliques. Trans. IPSJ, **44**, SIG14(TOM9), 57–70 (2003)
Johnson, D.S., Trick, M.A. (eds.): Cliques, coloring, and satisfiability. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26. American Mathematical Society Providence, RI (1996)
Kobayashi, S., Kondo, T., Okuda, K., Tomita, E.: (2003), Extracting globally structure free sequences by local structure freeness, In: Chen, J., Reif, J. (eds.): Proc. Ninth International Meeting on DNA Based Computers, p. 206 (2003)
Nakui, Y., Nishino, T., Tomita, E., Nakamura, T.: On the minimization of the quantum circuit depth based on a maximum clique with maximum vertex weight. Technical Report of RIMS, 1325, pp. 45–50. Kyoto University (2003)
Östergård, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Appl. Math. **120**, 197–207 (2002)
Pardalos, P.M., Xue J.: The maximum clique problem. J. Global Optim. **4**, 301–328 (1994)
Robson, J.M.: Algorithms for maximum independent sets. J. Algorithm, **7**, 425–440 (1986)
Sewell, E.C.: A branch and bound algorithm for the stability number of a sparse graph. INFORMS J. Comput. **10**, 438–447 (1998)
Stix, V.: Target-oriented branch and bound method for global optimization. J. Global Optim. **26**, 261–277 (2003)
Tarjan, R.E., Trojanowski, A.E.: Finding a maximum independent set. SIAM J. Comput. **6**, 537–546 (1977)
Tomita, E., Yamada, M.: An algorithm for finding a maximum complete subgraph. Conference Records of IECE (Technical Report of the National Convention of IECE 1978), p. 8 (1978)
Tomita, E., Kohata, Y., Takahashi, H.: A simple algorithm for finding a maximum clique. Technical Report of the University of Electro-Communications, UEC-TR-C5(1) (1988)
Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. Proc. Discrete Mathematics and Theoretical Computer Science. LNCS 2731, pp. 278–289 (2003)
Tomita, E., Kameda, T.: (revised 2006), An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. Technical Report of the University of Electro-Communications, UEC-TR-CAS10-2005 (2005)
Tomita, E., Nakagawa, T., Urabe, S.: An experimental comparison of algorithms for finding a maximum clique. Technical Report of The University of Electro-Communications, UEC-TR-CAS3-2006 (2006)
Tomita, E., Tanaka, A., Takahashi, H.: (to appear), The worst-case time complexity for generating all maximal cliques and computational experiments, Theoretical Computer Science (An invited paper in the Special Issue on COCOON 2004). The preliminary version was presented in: Tomita, E., Tanaka, A., Takahashi, H., The worst-case time complexity for generating all maximal cliques. Proc. International Computing and Combinatorics Conference (COCOON 2004), LNCS 3106. pp. 278–289.
Wood, D. R.: An algorithm for finding a maximum clique in a graph. Operations Res. Lett. **21**, 211–217 (1977)