A Much Faster Branch-and-Bound Algorithm for Finding a Maximum Clique

Etsuji Tomita[1(✉)], Kohei Yoshida[1], Takuro Hatta[1], Atsuki Nagao[1], Hiro Ito[1,2], and Mitsuo Wakatsuki[1]

[1] The University of Electro-Communications, Chofu, Tokyo 182-8585, Japan
`tomita@ice.uec.ac.jp`

[2] CREST, JST, Chiyoda, Tokyo 102-0076, Japan

Abstract. We present improvements to a branch-and-bound maximum-clique-finding algorithm MCS (WALCOM 2010, LNCS 5942, pp. 191–203) that was shown to be fast. First, we employ an efficient approximation algorithm for finding a maximum clique. Second, we make use of appropriate sorting of vertices only near the root of the search tree. Third, we employ a lightened approximate coloring mainly near the leaves of the search tree. A new algorithm obtained from MCS with the above improvements is named MCT. It is shown that MCT is much faster than MCS by extensive computational experiments. In particular, MCT is shown to be faster than MCS for gen400_p0.9_75 and gen400_p0.9_65 by over 328,000 and 77,000 times, respectively.

1 Introduction

We define a *clique* as a complete subgraph in which all pairs of vertices are adjacent to each other. Algorithms for finding a maximum clique (e.g., [18]) in a given graph have received much attention especially recently, since they have many applications. There has been much theoretical and experimental work on this problem [3,20]. In particular, while finding a maximum clique is a typical NP-hard problem, considerable progress has been made for solving this problem *in practice*. Furthermore, much faster algorithms are required in order to solve many practical problems. Along this line, Tomita et al. developed a series of branch-and-bound algorithms MCQ [16], MCR [17], and MCS [18] among others that run fast in practice. It was shown that MCS is relatively fast for many instances tested.

In this paper, we present improvements to MCS in order to make it much faster. First, we turn back to our original MCS [14] that employs an approximation algorithm for the maximum clique problem in order to obtain an initial lower bound on the size of a maximum clique. We choose here another approximation algorithm called *k-opt local search* [7] that runs quite fast. Second, we sort vertices as in MCR [17] and MCS [18] only appropriately near the root of the search tree. This technique is based on our successful earlier result [8]. Third, we employ lightened approximate coloring mainly near the leaves of the search

D. Zhu and S. Bereg (Eds.): FAW 2016, LNCS 9711, pp. 215–226, 2016.
DOI: 10.1007/978-3-319-39817-4_21

tree [8]. A new algorithm obtained from MCS with the above improvements is named MCT. It is shown that MCT is much faster than MCS by extensive computational experiments.

2 Definitions and Notation

We are concerned with a simple undirected graph $G = (V, E)$ with a finite set V of vertices and a finite set E of edges. The set V of vertices is considered to be *ordered*, and the i-th element in it is denoted by $V[i]$. A pair of vertices v and w are said to be adjacent if $(v, w) \in E$. For a vertex $v \in V$, let $\Gamma(v)$ be the set of all vertices that are adjacent to v in $G = (V, E)$. We call $|\Gamma(v)|$ the degree of v. For a subset $R \subseteq V$ of vertices, $G(R) = (R, E \cap (R \times R))$ is an *induced* subgraph. An induced subgraph $G(Q)$ is said to be a *clique* if $(v, w) \in E$ for all $v, w \in Q \subseteq V$, with $v \neq w$. In this case, we may simply say that Q is a clique. A largest clique in a graph is called a *maximum clique*, and the number of vertices in a maximum clique in $G(R)$ is denoted by $\omega(R)$.

3 Maximum Clique Algorithm MCS

3.1 Search Tree

The preceding branch-and-bound algorithm MCS [18] begins with a small clique and continues by finding larger and larger cliques. More precisely, we maintain global variables Q and Q_{max}, where Q consists of the vertices of the current clique and Q_{max} consists of the vertices of the largest clique found so far. Let $R \subseteq V$ consist of vertices (*candidates*) that can 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, add it to Q ($Q := Q \cup \{p\}$), and then compute $R_p := R \cap \Gamma(p)$ as the new set of *candidate* vertices. Such a procedure is represented by a *search tree*, where the root is V and, whenever $R_p := R \cap \Gamma(p)$ is applied then R_p is a child of R. The edge between R and $R_p := R \cap \Gamma(p)$ is called a *branch*.

3.2 Approximate Coloring: Numbering

In order to prune unnecessary searching, we used *greedy approximate coloring* or *Numbering* of the vertices in MCS. That is, each $p \in R$ is *sequentially* assigned a minimum possible positive integer value $No[p]$, called the *Number* or *Color* of p, such that $No[p] \neq No[r]$ if $(p, r) \in E$. Consequently, we have that $\omega(R) \leq \mathrm{M}ax\{No[p] | p \in R\}$. Hence, if $|Q| + \mathrm{M}ax\{No[p] | p \in R\} \leq |Q_{max}|$ holds, we need not continue the search for R.

After *Numbers* (*Colors*) are assigned to all vertices in R, we sort the vertices in nondecreasing order with respect to their *Numbers*. Vertices are expanded for searching from the rightmost to the leftmost on this R. Let $\mathrm{M}ax\{No[r] | r \in R\} = maxno$ and $C_i = \{r \in R | No[r] = i\}$, where $i = 1, 2, \ldots, maxno$.

3.3 Re-NUMBER

Because of the *bounding condition* mentioned above, if $No[r] \leq |Q_{max}| - |Q|$, then it is not necessary to search from vertex r. When we encounter a vertex p with $No[p] > |Q_{max}| - |Q|$, we attempt to change its $Number$ by **Procedure** Re-NUMBER described in Fig. 1, where No_p denotes the original value of $No[p]$ and $No_{th} := |Q_{max}| - |Q|$ stands for $No_{threshold}$. Try to find a vertex q in $\Gamma(p)$ such that $No[q] = k_1 \leq No_{th} - 1$, with $|C_{k_1}| = 1$. If such q is found, then try to find NUMBER k_2 such that no vertex in $\Gamma(q)$ has $Number$ k_2. If such number k_2 is found, then exchange the NUMBERs of q and p so that $No[q] = k_2$ and $No[p] = k_1$. When the vertex q with NUMBER k_2 is found in Fig. 1, $No[p]$ is changed from No_p to k_1 ($\leq No_{th} - 1$); thus, *it is no longer necessary to search from* p.

Procedure. Re-NUMBER was first proposed in MCS [14] and is shown to be quite effective [14,18,19].

```
procedure Re-NUMBER($p, No_p,$
            $No, C_1, C_2, ..., C_{maxno}$)
begin
  $No_{th} := |Q_{max}| - |Q|$;
  for $k_1 := 1$ to $No_{th} - 1$ do
    if $|C_{k_1} \cap \Gamma(p)| = 1$ then
      $q :=$ the element in $(C_{k_1} \cap \Gamma(p))$ ;
      for $k_2 := k_1 + 1$ to $No_{th}$ do
        if $C_{k_2} \cap \Gamma(q) = \emptyset$ then
          {Exchange the $Numbers$
                     of $p$ and $q$.}
          $C_{No_p} := C_{No_p} - \{p\}$;
          $C_{k_1} := (C_{k_1} - \{q\}) \cup \{p\}$;
          $No[p] := k_1$;
          $C_{k_2} := C_{k_2} \cup \{q\}$;
          $No[q] := k_2$;
          return
        fi od fi
  od
end { of Re-NUMBER}
```

Fig. 1. Procedure Re-NUMBER

3.4 EXTENDED INITIAL SORT-NUMBER

At the beginning of MCR and MCS, vertices are sorted in nondecreasing order from the rightmost to the leftmost mainly with respect to their degrees [17,18]. In addition, vertices are assigned initial $Numbers$. More precisely, the steps from {SORT} to just above EXPAND(V, No) in Fig. 4 (Algorithm MCR) in [17] is named *EXTENDED INITIAL SORT-NUMBER* to V. Note that *global variable* Q_{max} can be updated by "**then** $Q_{max} := R_{min}$" at the final stage of Fig. 4 (Algorithm MCR) in [17].

Here, MCS introduced another new *adjunct ordered set* V_a of vertices in order to preserve the order of the vertices sorted by EXTENDED INITIAL SORT-NUMBER. Approximate coloring is carried out in the order of V_a from the left to the right. Lastly, we reconstruct the adjacency matrix in MCS just after EXTENDED INITIAL SORT-NUMBER. This is to establish a more effective use of the cache memory.

The algorithm obtained as above is named MCS [18,19].

4 Improved Algorithms

4.1 Effective Use of an Approximate Solution

When the algorithm MCS was first proposed in [14], the first part of MCS consisted of a procedure for finding an approximately maximum clique of the given graph. Its approximation algorithm named init-lb [14] is a local search algorithm based on our previous work [15]. It finds a near-maximum clique in a very short time, and the result is used as an <u>init</u>ial <u>l</u>ower <u>b</u>ound of the size of a maximum clique. It demonstrated the effectiveness of an approximate solution for finding an exactly maximum clique. More precisely, when a sufficiently large near-maximum clique Q'_{max} is found, we let $Q_{max} := Q'_{max}$ at the beginning of MCS. Then $No_{th} := |Q_{max}| - |Q|$ becomes large and the bounding condition becomes more effective.

The final version of MCS presented in [18,19] excluded a procedure for finding an approximately maximum clique. This is because it is important to examine the performance of the main body of MCS [18] itself independently of an approximation algorithm. Batsyn et al. [1] and Maslov et al. [12] also demonstrated the effectiveness of an approximate solution, independently.

We have many approximation algorithms for finding a maximum clique [20], while finding a good approximate solution for the maximum clique problem is considered to be very hard [21]. The most important problem is a proper choice of the trade-off between the quality of the approximate solution and the time required to obtain it. We now turn back to our original MCS in [14] and choose another approximation algorithm called *k-opt local search* [7]. It does not necessarily give the best quality solution, but it runs quite fast and it is easy to control the above trade-off. The *k-opt local search* repeats a number of local searches from different vertices of the given graph. In this repetition, we select a vertex with the largest degree one by one from the sorted vertices with respect to their degrees by EXTENDED INITIAL SORT-NUMBER. When the number of repetitions becomes large, the quality of the solution increases but with increased running time.

In order to give a proper compromise between the high quality of the solution and the time required to obtain it for the given graph $G = (V, E)$ with $n = |V|, m = |E|$, and $dens = 2m/n(n-1)$ (*density*), we have chosen the number *rep* of *repetitions* as follows by preliminary experiments:

$$rep = \min\{20n^{1/2} \times dens^3,\ n\} \quad for \ \ n \geq 1.$$

Hereafter, a procedure for finding an approximate maximum clique of the given graph $G = (V, E)$ under the above condition is named KLS(V, Q'_{max}) and its solution is given to Q'_{max}.

The new MCS that is composed of a combination of the KLS procedure and MCS in [18] as above is named MCS$_1$.

4.2 EXTENDED INITIAL SORT-NUMBER Near the Root of the Search Tree

It is shown that both search space and overall running time are reduced when vertices are sorted in a nondecreasing order with respect to their degrees prior to application of a branch-and-bound depth-first search for finding a maximum clique [4,5,15,16]. All of the preceding algorithms MCQ, MCR, and MCS employ such sorting of vertices at the root level ($depth = 0$) of the search trees. It is also made clear that if the vertices are sorted as above and followed by $Numbering$ at every depth of the search tree then the resulting search space becomes more reduced but with much more overhead of time [8].

Therefore, it becomes important to choose a good trade-off between the reduction of the search space and the time to realize it. For an earlier algorithm MCLIQ [15] that is a predecessor of MCQ, we proposed a technique to solve the above trade-off and reduced the overall running time successfully in the way as follows [8]:

(i) At the first stage near the root of the search tree, we apply sorting of vertices followed by $Numbering$. ([8])
(ii) In the second stage of the search tree, we apply $Numbering$ without new sorting of vertices. (Just as in [15])
(iii) In the third stage of the search tree near the leaves, we expand vertices by only inheriting the order of vertices and the previous NUMBERs. (Just as in [5]).

The above techniques are considered to be promising for any algorithm for finding a maximum clique if we control these three stages appropriately. So, we apply the techniques of [8] to MCS. Here, we make full use of the adjunct ordered set V_a of vertices in MCS [18] in which vertices are sorted in nondecreasing order with respect to their degrees from the rightmost (end) to the leftmost (front) by EXTENDED INITIAL SORT-NUMBER in [18]. In addition, we avoid the set R of vertices in MCS [18] so that we are free from the task of reconstructing such R in which vertices are sorted with respect to their NUMBERs. From now on, we rename V_a as R, for simplicity. So, be careful that the set R in this paper corresponds to V_a, and not to R in MCS [18].

Hereafter, the NUMBERing procedure combined with Re-NUMBER is named NUMBER-R and is shown in Fig. 2. This is exactly the first half of the **procedure** Re-NUMBER-SORT in Fig. 2 of MCS [18].

A slightly stronger procedure Re-NUMBER1 is defined as the one obtained from **procedure** Re-NUMBER by replacing "**for** $k_2 := k_1 + 1$ **to** No_{th} **do**" by "**for** $k_2 := 1$ **to** $k_1 - 1$ **and** $k_1 + 1$ **to** No_{th} **do**". Another slightly modified **procedure** NUMBER-R+(R, No) is defined as the one obtained from **procedure** NUMBER-R(R, No) by replacing "**if** $(k > No_{th})$ **and** $(k = maxno)$ **then**" by "**if** $(k > No_{th})$ **then**" and "Re-NUMBER-R" by "Re-NUMBER1" in NUMBER-R(R, No). That is, the condition for applying Re-NUMBER is relaxed in **procedure** NUMBER-R+(R, No).

```
procedure NUMBER-R($R, No$)
begin
  {NUMBER}
    $maxno := 0$;
    $C_1 := \emptyset$;
    for $i := 1$ to $|R|$ do
      { Conventional greedy
        approximate coloring }
      $p := R[i]$ ;
      $k := 1$;
      while $C_k \cap \Gamma(p) \neq \emptyset$
        do $k := k + 1$ od
      if $k > maxno$ then
        $maxno := k$;
        $C_{maxno} := \emptyset$
      fi
      $C_k := C_k \cup \{p\}$;
      $No[p] := k$;

      { - Re-NUMBER starts - }
      $No_{th} := |Q_{max}| - |Q|$;
      if $(k > No_{th})$ and
          $(k = maxno)$ then
        Re-NUMBER($p, k, No,$
            $C_1, C_2, ..., C_{maxno}$) ;
        if $C_{maxno} = \emptyset$ then
          $maxno := maxno - 1$
        fi
      fi
      { - Re-NUMBER ends - }

    od
end { of NUMBER-R }
```

Fig. 2. Procedure NUMBER-R

```
procedure NUMBER-RL($R, No, newNo$)
begin
  $No_{th} := |Q_{max}| - |Q|$;
  for $i := 1$ to $|R|$ do
    $C_i := \emptyset$;
  od
  $maxno := 1$;
  for $i := 1$ to $|R|$ do
    if $No[R[i]] \leq No_{th}$ then
      $k := No[R[i]]$;
      if $k > maxno$ then $maxno := k$ fi
      $C_k := C_k \cup \{R[i]\}$; $newNo[R[i]] := k$;
    fi
  od
  for $i := 1$ to $|R|$ do
    if $No[R[i]] > No_{th}$ then
      $p := R[i]$ ;   $k := 1$;
      while $C_k \cap \Gamma(p) \neq \emptyset$
        do $k := k + 1$ od
      if $k > maxno$ then
        $maxno := k$;
      fi
      $C_k := C_k \cup \{p\}$;
      $newNo[p] := k$;
      if $(k > No_{th})$ then
        Re-NUMBER1($p, k, No,$
          $C_1, C_2, \ldots, C_{maxno}$) ;
        if $C_{maxno} = \emptyset$ then
          $maxno := maxno - 1$
        fi
      fi
    fi
  od
end { of NUMBER-RL}
```

Fig. 3. Procedure NUMBER-RL

At the first stage near and including the root of the search tree, we sort a set of vertices by EXTENDED INITIAL SORT-NUMBER to R followed by Numbering by NUMBER-R+(R, No). The procedure is shown in Fig. 4 with "$Th_1 = 0.4, Th_2 = 0$" instead of "$Th_1 = 0.4, Th_2 = 0.1$" at {Switches}. It is experimentally confirmed that NUMBER-R+(R, No) is better than NUMBER-R(R, No), since NUMBER-R+(R, No) is applied only a few times with better results but with more overhead than NUMBER-R(R, No).

This task of preprocessing (of sorting vertices followed by NUMBER-R) is time-consuming. So, as stated at the beginning of Sect. 4.2, it is important to change this first stage to the second stage at an appropriate switching depth that is near the root of the search tree. First, for a vertex $p \in R$ at a certain depth of the search tree, consider $newR := R_p = R \cap \Gamma(p)$ that is a child of R. If the ratio $|\{v|No[v] > No_{th}\}|/|newR|$ becomes large, it is considered that much more preprocessing becomes appropriate. In addition, when *dens* (density) of the graph becomes larger it generally requires more time for finding a maximum clique and then much more number of preprocessing becomes appropriate. As a result, we consider the following value:

$$T = \frac{|\{v|No[v] > No_{th}\}|}{|newR|} \times dens.$$

From preliminary experiments, we have chosen that if $T \geq 0.4$ then we continue the same procedure described for the first stage. Otherwise, we switch the stage to the second stage. Thus, we let $Th_1 := 0.4$ in Fig. 4. The new procedure obtained from Fig. 4 by replacing "$Th_1 := 0.4, Th_2 := 0.1$" by "$Th_1 := 0.4, Th_2 := 0$" at {Switches} is named MCS$_2$. Here, we control the $stage = 1$ so that it never returns back to $stage = 1$ after it changed to the second or the third $stage (\neq 1)$. Konc and Janežič [9] also improved MCQ [16] successfully in a similar way as in [8], independently.

4.3 Lightened Numbering Mainly Near the Leaves of the Search Tree

Mainly near the leaves of the search tree, the ratio $|\{v|No[v] > No_{th}\}|/|newR|$ tends to be small. In this third stage, it is preferable to lighten the task of preprocessing before expansion of vertices. So, we only inherit the order of vertices from that in their parent depth, as in the second stage. In addition, we inherit the assigned NUMBERs from those assigned to their parents only if their NUMBERs are less than or equal to No_{th}. If we inherit all the assigned NUMBERs from those assigned to their parents as in [5] the resulting bounding condition becomes too weak. In order to remedy this weakness, if the inherited NUMBERs from those assigned to their parents are greater than No_{th} then we give them new NUMBERs. For vertices whose inherited NUMBERs from their parents are greater than No_{th} we newly give them NUMBERs by sequential numbering combined with Re-Numbering. For this Re-Numbering we adopt stronger Re-NUMBER1 instead of Re-NUMBER since Re-Numbering is required not so many times in this stage. The resulting procedure in this stage named **procedure** NUMBER-RL is shown in Fig. 3.

From preliminary experiments, we have chosen to turn to the new $stage = 3$ if the previously given value $T = (|\{v|No[v] > No_{th}\}|/|newR|) \times dens$ is less than 0.1. Then we let $Th_2 := 0.1$ in Fig. 4. The **procedure** NUMBER-RL is weaker than the previous **procedure** NUMBER-R for obtaining strong bounding condition, but it requires less overhead than the previous one. However, if the given graph is too dense then **procedure** NUMBER-RL becomes too weak and the number of branches of the search tree grows quite large. So, we choose to go to new $stage = 3$ only if $dens \leq 0.95$. In addition, a simpler algorithm is generally better than sophisticated algorithms for sparse graphs.

```
procedure MCT($G = (V, E)$)
begin
  global $Q := \emptyset$;
  global $Q_{max} := \emptyset$;
  global $dens := 2|E|/|V|(|V| - 1)$;
          {density}
  if $dens \leq 0.1$ then
     MCS($G = (V, E)$);
  else
    $Th_1 := 0.4$;   $Th_2 := 0.1$;
       {Switches}
    Apply EXTENDED INITIAL
             SORT-NUMBER to $V$;
       {$Q_{max}$ can be updated.}
    Reconstruct the adjacency
         matrix as described in [18];
    KLS($V, Q'_{max}$);
    if $Q_{max} < Q'_{max}$ then
    $Q_{max} := Q'_{max}$    fi
    NUMBER-R+($V, No$);
       $stage := 1$;
    EXPAND ($V, No, stage, Th_1, Th_2$);
  fi
 output $Q_{max}$ {Maximum clique}
end { of MCT}
```

Fig. 4. Procedure MCT

```
procedure EXPAND($R, No, stage, Th_1, Th_2$)
begin
 for $i := |R|$ downto 1 do
   $p := R[i]$;
   if ($stage = 1$ and $|Q| + \max_{v \in R}\{No[v]\} > |Q_{max}|$)
   or ($stage \neq 1$ and $|Q| + No[p] > |Q_{max}|$) then
     $Q := Q \cup \{p\}$;
     $newR := R \cap \Gamma(p)$;   {preserving the order}
     if $newR \neq \emptyset$ then
       $No_{th} := |Q_{max}| - |Q|$;
       $T := \frac{|\{v | No[v] > No_{th}\}|}{|newR|} \times dens$;
       if $stage = 1$ and $Th_1 \leq T$ then
         Apply EXTENDED INITIAL
               SORT-NUMBER to $R$;
         NUMBER-R+($newR, newNo$);
         {The initial value of $newNo$ has no significance.}
         $newstage := 1$;
       else if $dens > 0.95$ or $Th_2 \leq T$ then
         NUMBER-R($newR, newNo$);
         $newstage := 2$;
       else
         NUMBER-RL($newR, No, newNo$);
         $newstage := 3$;
       fi
       EXPAND($newR, newNo, newstage, Th_1, Th_2$)
     else if $|Q| > |Q_{max}|$ then $Q_{max} := Q$ fi
     fi
     $Q := Q - \{p\}$;
     $R := R - \{p\}$;    {preserving the order}
   fi
 od
end { of EXPAND }
```

Fig. 5. Procedure EXPAND

So, if $dens \leq 0.1$ we choose simpler algorithm MCS [18] without relying on any new technique introduced in this paper (Fig. 4).

The resulting algorithm obtained by taking the total techniques in Sects. 4.1, 4.2 and 4.3 to improve MCS [18] is named MCT (The 'T' is for 'Total'.) and is shown in Fig. 4.

5 Computational Experiments

In order to demonstrate the effectiveness of the techniques given in the previous section, we carried out computational experiments. All the algorithms were implemented in C language. The computer had an Intel core i7-4790 CPU of 3.6 GHz clock with 8 GB of RAM and 8 MB of cache memory. It worked on a Linux operating system with a compiler gcc -O3. The *dfmax running time for DIMACS benchmark instances* [6] for `r300.5`, `r400.5` and `r500.5` are 0.14, 0.90 and 3.44 seconds, respectively.

5.1 Stepwise Improvement

Table 1 shows stepwise improvement from MCS to MCT for selected graphs chosen from the next Table 2.

Table 1. Comparison of MCS, MCS$_1$, MCS$_2$ and MCT

Graph	Times [sec]				Branches [$\times 10^{-6}$]			
	MCS	MCS$_1$	MCS$_2$	MCT	MCS	MCS$_1$	MCS$_2$	MCT
brock400_1	288	256	182	116	89	77	52	55
brock800_4	1,768	1,751	1,256	819	381	380	258	270
C250.9	1,171	926	774	404	255	197	154	186
gen400_p0.9_55	22,536	1,651	1,970	167	2,895	181	210	61
gen400_p0.9_65	57,385	5.73	6.07	0.74	7,628	0.33	0.34	0.13
gen400_p0.9_75	108,298	1.38	1.38	0.33	17,153	0.05	0.05	0.02
p_hat700-3	900	456	438	216	88	43	40	54
p_hat1000-2	85	47	46	29	13	6.6	6.3	10
p_hat1500-2	6,299	2,964	2,832	1,560	560	253	234	400
san400_0.7_1	0.26	0.06	0.06	0.06	22,771	200	0	0
frb-30-15-2	1,048	691	773	116	229	135	148	61

(1) Improvement from MCS to **MCS$_1$** by an approximate solution in Sect. 4.1: The improvement is particularly quite effective for the gen graph family. MCS$_1$ is faster than MCS for gen400_p0.9_75 and gen400_p0.9_65 by more than 78,000 and 10,000 times, respectively. This technique is effective for almost all graphs but with few exceptions as for the MANN graph family.
(2) Improvement from MCS$_1$ to **MCS$_2$** by EXTENDED INITIAL SORT-NUMBER in Sect. 4.2: This technique is effective mainly for the brock graph family by around 1.4 times. For some graphs such as the gen and frb graph families, the effect is negative.
(3) Improvement from MCS$_2$ to **MCT** by Lightened Numbering in Sect. 4.3: This technique is effective for almost all graphs in reducing computing time in spite of increased numbers of branches in general. MCT is faster than MCS$_2$ for gen400_p0.9_55 and gen400_p0.9_65 by more than 11 and 8 times, respectively, where their numbers of branches are also reduced.

5.2 Overall Improvement

Table 2 shows the result of the overall improvement from MCS to MCT in computing time for the benchmark graphs where the columns *sol* and *time* below KLS show the solution and the computing time by KLS, respectively. The benchmark graphs include brock - DSJ graphs in DIMACS [6] and the frb family in BHOSLIB [2]. They also include random graphs of r200.8 - r20000.1 where r$n.p$ stands for a random graph with the number of vertices $= n$ and the edge probability $= p$. The averages are taken over 10 random graphs except for r200.9 and r200.95 whose averages are taken over 100 random graphs. The state-of-the-art result of BBMCX (MCX *for short*) [13] by Segundo et al. is included. Here, its computing time is calibrated on the established way in the Second DIMACS Implementation Challenge [6], where our computer is calculated to be 1.30 times

Table 2. CPU time [sec] for benchmark graphs

Graph				KLS		MCS	MCT	MCX	MaxC	I&M	BG14
Name	n	*dens*	ω	*sol*	*time*	[18]		[13]	[11]	[12]	[1]
brock200_1	200	0.75	21	21	0.01	0.36	0.23	**0.18**	0.34	4.41	2.51
brock400_1	400	0.75	27	25	0.08	288	**116**	150	205	188	302
brock400_2	400	0.75	29	24	0.08	124	**52**	68	96	94	132
brock400_3	400	0.75	31	24	0.08	195	**86**	120	160	145	211
brock400_4	400	0.75	33	25	0.08	103	**46**	68	100	72	87
brock800_1	800	0.65	23	21	0.22	4,122	**1,950**	2,690	4,560	4,000	4,220
brock800_2	800	0.65	24	21	0.22	3,683	**1,630**	2,420	4,000	3,460	3,780
brock800_3	800	0.65	25	21	0.22	2,540	**1,110**	1,590	2,510	2,360	2,650
brock800_4	800	0.65	26	20	0.22	1,768	**819**	1,100	1,850	1,680	1,870
C250.9	250	0.90	44	44	0.08	1,171	404	713	**268**		
C2000.5	2000	0.50	16	15	0.59	33,899	**21,027**				
gen200_p0.9_44	200	0.90	44	44	0.05	0.174	**0.076**	0.155	0.115	1.68	
gen200_p0.9_55	200	0.90	55	55	0.06	0.458	**0.068**	0.312	0.142	2.43	0.917
gen400_p0.9_55	400	0.90	55	53	0.25	22,536	**167**	19,400		46,500	2,960
gen400_p0.9_65	400	0.90	65	65	0.26	57,385	**0.74**	66,100	36,700	2,130	19
gen400_p0.9_75	400	0.90	75	75	0.28	108,298	**0.33**	47,200	9,980	83.5	7.8
MANN_a27	378	0.99	126	126	0.81	0.26	1.05	0.18	**0.16**	1.30	
MANN_a45	1035	0.99	345	344	21.5	53.4	75.5	32.0	22.7	**17.3**	55.1
p_hat300-3	300	0.74	36	36	0.06	0.99	**0.28**	0.66	1.16	6.72	3.62
p_hat500-3	500	0.75	50	50	0.22	57.1	**17.4**	33.3	39.6	50.3	59.5
p_hat700-3	700	0.75	62	62	0.46	900	**216**	680	879	552	767
p_hat1000-2	1000	0.49	46	46	0.23	85	**29**	73	101	204	113
p_hat1000-3	1000	0.74	68	68	1.00	305,146	**38,800**				
p_hat1500-1	1500	0.25	12	11	0.03	1.8	**1.4**	2.0	10	478	422
p_hat1500-2	1500	0.51	65	65	0.73	6,299	**1,560**	3,850	8,030	5,350	5,430
san1000	1000	0.50	15	10	0.06	1.02	**0.21**	0.68	0.72	449	158
san200_0.7_1	200	0.70	30	30	0.01	**0.0037**	0.0133	0.0115	0.0092	7.62	
san200_0.9_1	200	0.90	70	70	0.07	0.0848	0.0727	0.0385	**0.0131**	1.35	
san400_0.7_1	400	0.70	40	40	0.06	0.26	**0.06**	0.14	0.13	15.80	6.69
san400_0.7_2	400	0.70	30	30	0.05	0.0589	**0.0519**	0.0923	0.0638	19.3	
san400_0.7_3	400	0.70	22	18	0.05	0.665	**0.273**	0.391	0.433	26.9	11.6
sanr200_0.7	200	0.70	18	18	0.01	0.15	0.11	**0.079**	0.17	5.05	1.03
sanr200_0.9	200	0.90	42	42	0.06	15.3	4.67	7.38	**4.21**	4.62	10.2
sanr400_0.5	400	0.50	13	13	0.01	0.351	0.274	**0.186**	0.688	34.9	17.6
sanr400_0.7	400	0.70	21	21	0.06	77.3	**40.7**	44.5	81.2	86.2	81.4
DSJC500.5	500	0.50	13	13	0.02	1.53	1.20	**0.81**	2.84		
DSJC1000.5	1000	0.50	15	15	0.12	141	**93**	102	265		
keller5	776	0.75	27	27	0.34	82,421	10,000	30,300	**4,980**	5,780	82,500
frb30-15-1	450	0.82	30	28	0.15	740	**156**	1,029	560		
frb30-15-2	450	0.82	30	30	0.15	1,048	**116**	672	758		
frb30-15-3	450	0.82	30	28	0.15	670	**124**	350	477		
frb30-15-4	450	0.82	30	28	0.15	2,248	**535**	1,157	955		
frb30-15-5	450	0.82	30	28	0.15	972	**156**	801	705		
r200.8	200	0.8	24-27	24-27	0.028	1.66	**0.78**	0.95	1.08		
r200.9	200	0.9	39-43	39-43	0.060	27.0	10.7	14.8	**6.2**		
r200.95	200	0.95	58-66	58-66	0.098	21.1	10.3	30.2	**2.5**		
r500.6	500	0.6	17-18	16-17	0.056	18	11	**10**	22		
r500.7	500	0.7	22-23	21-22	0.101	723	**340**	423	564		
r1000.4	1000	0.4	12	11	0.045	5.99	5.14	**4.52**	14.5		
r1000.5	1000	0.5	15-16	14-15	0.122	134	**92**	103	231		
r5000.1	5000	0.1	7	5-6	0.149	**1.17**	**1.17**	1.19	68		
r5000.2	5000	0.2	9-10	7-8	0.21	45	**39**	68	78		
r5000.3	5000	0.3	12	10-11	0.52	2,283	**1,875**				
r10000.1	10000	0.1	7	5-6	0.58	**14**	**14**	20	684		
r10000.2	10000	0.2	10	8-9	0.87	1,303	**1,139**				
r15000.1	15000	0.1	8	6	1.30	**62**	**62**	114	2,749		
r20000.1	20000	0.1	8	6-7	2.31	**234**	**234**				

faster than that in [13]. The calibrated computing time of MaxCLQ (MaxC *for short*) [10,11] by Li and Quan is also included from [13]. The calibrated computing time by ILS&MCS (I&M *for short*) [12] and BG14 [1] are added on the assumption that the performance of each MCS is the same, for reference, too. The boldface entries indicate the fastest time in the row.

The result shows that MCT is faster than MCS for graphs gen400_p0.9_75, gen400_p0.9_65, gen400_p0.9_55, frb-30-15-2, keller5, p_hat1000-3, gen200_p0.9_55, frb-30-15-5 and frb-30-15-3 by over 328,000, 77,000, 134, 9.0, 8.2, 7.8, 6.7, 6.2 and 5.4 times, respectively. MCT is faster than MCS for graphs san1000, frb-30-15-1, san400_0.7_1, frb-30-15-4, p_hat700-3 and p_hat1500-2 by over 4 times, and for graphs p_hat300-3, p_hat500-3 and sanr200_0.9 by over 3 times. In Table 2, MCT is faster than MCS by more than 2 times for the other 16 graphs including r200.9, r200.8, r500.7 and r200.95. Except for few special graphs as in MANN family and for easy graphs that can be solved in a very short time, MCT is faster than MCS for almost all graphs in the instances tested.

MCT is faster than the other algorithms in Table 2 for many instances. Note that MaxCLQ (MaxC) is fast for dense graphs.

In conclusion, MCT has achieved significant improvement over MCS, that is, MCT is much faster than MCS.

Acknowledgements. We express our sincere gratitude to the referees, E. Harley, and T. Toda for their useful comments and help. This research was supported in part by MEXT&JSPS KAKENHI Grants, JST CREST grant and Kayamori Foundation grant.

References

1. Batsyn, M., Goldengorin, B., Maslov, E., Pardalos, P.M.: Improvements to MCS algorithm for the maximum clique problem. J. Comb. Optim. **27**, 397–416 (2014)
2. http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm
3. 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, Boston (1999)
4. Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem. Oper. Res. Lett. **9**, 375–382 (1990)
5. Fujii, T., Tomita, E.: On efficient algorithms for finding a maximum clique, Technical report of IECE, AL81-113, 25–34 (1982)
6. Johnson, D.S., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiability. DIMACS Series in DMTCS, vol. 26. American Mathematical Society, Boston (1996)
7. Katayama, K., Hamamoto, A., Narihisa, H.: An effective local search for the maximum clique problem. Inf. Process. Lett. **95**, 503–511 (2005)
8. Kohata, Y., Nishijima, T., Tomita, E., Fujihashi, C., Takahashi, H.: Efficient algorithms for finding a maximum clique, Technical report of IEICE, COMP89-113, 1–8 (1990)
9. Konc, J., Janežič, D.: An improved branch and bound algorithm for the maximum clique problem. MATCH Commun. Math. Comput. Chem. **58**, 569–590 (2007)
10. Li, C.M., Quan, Z.: An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In: AAAI Conference on AI, pp. 128–133 (2010)

11. Li, C.M., Quan, Z.: Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: Proceedings of the IEEE ICTAI, pp. 344–351 (2010)
12. Maslov, E., Batsyn, M., Pardalos, P.M.: Speeding up branch and bound algorithms for solving the maximum clique problem. J. Glob. Optim. **59**, 1–21 (2014)
13. Segundo, P.S., Nikolaev, A., Batsyn, M.: Infra-chromatic bound for exact maximum clique search. Comput. Oper. Res. **64**, 293–303 (2015)
14. Sutani, Y., Higashi, T., Tomita, E., Takahashi, S., Nakatani, H.: A faster branch-and-bound algorithm for finding a maximum clique, Technical report of IPSJ, 2006-AL-108, 79–86 (2006)
15. Tomita, E., Kohata, Y., Takahashi, H.: A simple algorithm for finding a maximum clique, Technical report of the Univ. of Electro-Commun., UEC-TR-C5(1) (1988)
16. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731, pp. 278–289. Springer, Heidelberg (2003)
17. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Glob. Optim. **37**, 95–111 (2007)
18. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: Rahman, M.S., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 191–203. Springer, Heidelberg (2010)
19. Tomita, E., Sutani, Y., Higashi, T., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments. IEICE Trans. Inf. Syst. **E96–D**, 1286–1298 (2013)
20. Wu, Q., Hao, J.K.: A review on algorithms for maximum clique problems. Eur. J. Oper. Res. **242**, 693–709 (2015)
21. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of the STOC 2006, pp. 681–690 (2006)