Electronic Preprint for Journal of Information Processing Vol.25

Regular Paper

# A Much Faster Algorithm for Finding a Maximum Clique with Computational Experiments

Etsuji Tomita[1,a)] Sora Matsuzaki[1,b)] Atsuki Nagao[2,c)] Hiro Ito[1,3,d)] Mitsuo Wakatsuki[1,e)]

Received: November 8, 2016, Accepted: February 9, 2017

**Abstract:** We present further 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 a variant of an efficient approximation algorithm KLS 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 $_{k5}$MCT. It is shown that $_{k5}$MCT is much faster than MCS by extensive computational experiments. In particular, $_{k5}$MCT is shown to be faster than MCS for gen400_p0.9_75, gen400_p0.9_65 and gen400_p0.9_55 by over 81,000, 39,000 and 19,000 times, respectively.

**Keywords:** maximum clique, branch-and-bound algorithm, bounding condition, computational experiments

## 1. Introduction

Given an undirected graph $G$, a *clique* is defined to be a complete subgraph of $G$ in which all pairs of vertices are adjacent to each other. Finding a *maximum* clique, that is a clique of maximum cardinality, is a typical NP-hard problem [4]. So, it is very difficult to obtain the exact solution of this problem in general. In addition, it is also difficult to obtain even a satisfactory approximate solution [38]. Finding a maximum independent set in a graph is also equivalent to finding a maximum clique of its complementary graph. Here, finding a maximum clique in a graph has many important practical applications. These applications include coding theory [24], pattern recognition and image processing [9], [20], bioinformatics [5], [18], [33], design of wireless networks [16], and others [4], [36], [37].

Hence, much effort has been devoted to this problem theoretically and experimentally [4], [21], [37]. In particular, see [37] for a recent progress of algorithms for this problem. Furthermore, much faster algorithms are in great demand so that much more practical problems can be solved. Along this line, Tomita et al. developed a series of branch-and-bound algorithms MCQ [29], MCR [31] and MCS [32], [34] among others that run fast in practice. It was experimentally shown that MCS is relatively fast for many instances tested.

In this paper, we present improvements to MCS to make it much faster. First, we turn back to our original MCS [25] that employs an approximation algorithm for the maximum clique problem at the beginning in order to obtain an initial lower bound on the size of a maximum clique, as noted at Concluding Remarks in Ref. [34]. We choose here another approximation algorithm called *k-opt local search* (KLS for short) by Katayama et al. [11] that runs in quite a short time. Second, we sort vertices as in MCR [31] and MCS [32] only appropriately near the root of the search tree. This technique is based on our successful earlier results [12], [19], [23]. Third, we employ lightened approximate coloring mainly near the leaves of the search tree [12]. A new algorithm obtained from MCS with the above improvements is named $_{k5}$MCT. It is shown that $_{k5}$MCT is much faster than MCS by extensive computational experiments.

The preliminary versions of this paper appeared in Refs. [8] and [35]. This paper is an extended version of Ref. [35] with a slight modification.

## 2. Definitions and Notation

(1) 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 that comprise *unordered* pairs $(v, w)$ $(= (w, v))$ of distinct vertices. The set $V$ of vertices is considered to be *ordered*, and the $i$-th 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 vertex $v \in V$, let $\Gamma(v)$ be the set of all vertices that are adjacent to $v$ in $G = (V, E)$, i.e., $\Gamma(v) = \{w \in V \mid (v, w) \in E\}$. We call $|\Gamma(v)|$, the number of vertices adjacent to a vertex $v$, the *degree* of $v$. Here, the number of elements in a set $S$ is denoted by $|S|$.

(3) 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,

[1] The University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[2] Seikei University, Musashino, Tokyo 180–8633, Japan
[3] CREST, Japan Science and Technology Agency, Chiyoda, Tokyo 102–0076, Japan
[a)] tomita@ice.uec.ac.jp
[b)] m1311182@edu.cc.uec.ac.jp
[c)] a-nagao@st.seikei.ac.jp
[d)] itohiro@uec.ac.jp
[e)] wakatsuki.mitsuo@uec.ac.jp

we may simply say that $Q$ is a clique. In particular, a clique which is not properly contained in any other clique is called *maximal*. A maximal clique of the maximum cardinality in a graph is called a *maximum clique*, and the number of vertices in a maximum clique in $G(R)$ is denoted by $\omega(R)$.

An *independent set* is defined to be a subgraph in which any pair of vertices are not adjacent to each other.

## 3. Maximum Clique Algorithm MCS

### 3.1 The Branch-and-Bound Algorithm

One standard approach for finding a maximum clique is based upon the branch-and-bound depth-first search method. Each of the preceding algorithm MCQ [29], MCR [31], and MCS [32] is also a branch-and-bound one that begins with a small clique and continues by finding larger and larger cliques. To be precise, we maintain global variables $Q$ and $Q_{\max}$, where $Q = \{p_1, p_2, \ldots, p_d\}$ consists of the vertices of the current clique and $Q_{\max}$ consists of the vertices of the largest clique found so far. Let $R = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \cdots \cap \Gamma(p_d) \subseteq V$ consist of *candidate* vertices that can be added to $Q$ to enlarge $Q$. 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. This procedure is applied recursively while $R_p \neq \emptyset$.

When $R_p = \emptyset$ is reached, then $Q$ constitutes a *maximal* clique. If $Q$ is maximal and $|Q| > |Q_{\max}|$ holds, then $Q_{\max}$ 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 until $R = \emptyset$.

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

To make the above branch-and-bound algorithm efficient, it is most important to prune unnecessary searching with *low overhead*. For this purpose, we employed *greedy approximate coloring* or *Numbering* of the vertices in MCQ and MCR, as introduced in Refs. [7], [26], [27]. That is, each $p \in R$ is *sequentially* assigned a minimum possible positive integral 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 the following property.

**Proposition** Let $\chi(R)$ be the minimum possible number of colors to color a subgraph induced by $R$. Then,

$$\omega(R) \leq \chi(R) \leq \mathrm{Max}\{No[p] \mid p \in R\}. \qquad \square$$

Hence, if $|Q| + \mathrm{Max}\{No[p] \mid p \in R\} \leq |Q_{\max}|$ holds, we need not continue the search for $R$. This is a *bounding condition*.

At the beginning of MCQ, we sort vertices of $V$ in descending order with respect to their degrees. In MCR and MCS, vertices of $V$ are sorted in a similar but more sophisticated way.

After *Numbers* (*Colors*) are assigned to all vertices in $R$, let $\mathrm{Max}\{No[r] \mid r \in R\} = maxno$ and $C_i = \{r \in R \mid No[r] = i\}$, where $i = 1, 2, \ldots, maxno$, that is, $C_i$ is an independent set of vertices whose numbers are $i$. We sort the vertices in ascending order with respect to their *Numbers* so that $R = C_1 \cup C_2 \cup \cdots \cup C_{maxno}$ as an ordered set of vertices.

Vertices are expanded for searching from the rightmost (with the *largest Number*) to the leftmost (with the *smallest Number*) on this $R$.

### 3.3 New Approximate Coloring

In order to make the above bounding condition work more efficiently, the algorithm MCS [25] first introduced the following **procedure** Re-NUMBER.

Because of the bounding condition mentioned in Section 3.2, 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* to be less than or equal to $|Q_{\max}| - |Q|$ in the following manner. Let $No_p$ denote the original value of $No[p]$ and $No_{th} := |Q_{\max}| - |Q|$ stand for $No_{threshold}$. Attempt 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 attempt 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 *Number* of $q$ so that $No[q] = k_2$. (If no vertex $q$ with *Number* $k_2$ as above is found, then nothing is done.) When the *Number* of vertex $q$ is changed from $k_1$ to $k_2$, $No[p]$ is changed from $No_p$ to $k_1$ ($\leq No_{th} - 1$); thus, *it is no longer necessary to search from* $p$.

The above procedure is named "Re-NUMBER" and is de-

```
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_k1 ∩ Γ(p)| =1 then
   q := the element in (C_k1 ∩ Γ(p)) ;
   for k_2 := k_1 + 1 to No_th do
    if C_k2 ∩ Γ(q) = ∅ then
     {Exchange the Numbers of p and q.}
     C_k2 := C_k2 ∪ {q};
     No[q] := k_2;
     C_No_p := C_No_p − {p};
     C_k1 := (C_k1 − {q}) ∪ {p};
     No[p] := k_1;
     return
    fi
   od
  fi
 od
end { of Re-NUMBER}
```

**Fig. 1** Procedure Re-NUMBER.

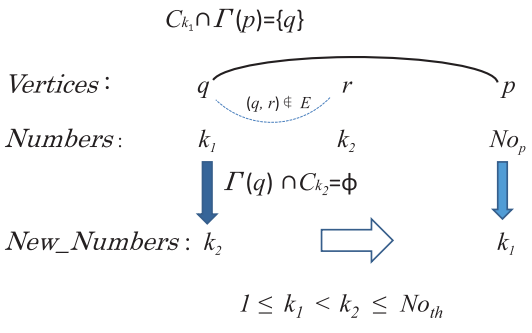


**Fig. 2** ReNumbering.

scribed in **Fig. 1**. See **Fig. 2** for an illustration. **Procedure** Re-NUMBER is a central part of MCS and is shown to be quite effective [25], [32], [34]. Some of its variations are also employed in this paper.

### 3.4 EXTENDED INITIAL SORT-NUMBER

At the beginning of MCR and MCS, the given set $V$ of $n$-vertices is sorted to $V = \{V[1], V[2], \ldots, V[n]\}$ so that a subgraph of $G = (V, E)$ induced by a set of vertices $\{V[1], V[2], \ldots, V[i]\}$, it holds that $V[i]$ aways has the minimum degree in $\{V[1], V[2], \ldots, V[i]\}$ for $1 \leq i \leq |V|$ as in Ref. [6]. Here, the degrees of their adjacent vertices are also taken into consideration. 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 Ref. [31] is named *EXTENDED INITIAL SORT-NUMBER* to $V$. Note that *global variable* $Q_{\max}$ can be updated by

**then** $Q_{\max} := R_{\min}$

at line 5 from the bottom of Fig. 4 (Algorithm MCR) in Ref. [31].

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. (See Fig. 4 in Ref. [34] for an illustration.)

### 3.5 Reconstruction of the Adjacency Matrix

Each graph is stored as an adjacency matrix in the computer memory. Sequential numbering is carried out according to the initial order of vertices in the adjunct ordered set $V_a$. Taking this into account, we *rename* the vertices of the graph and *reconstruct* the adjacency matrix so that the vertices are *consecutively ordered* in a manner identical to *the initial order of vertices* obtained at the beginning. (See Fig. 5 in Ref. [34] for an illustration.) The above-mentioned reconstruction of the adjacency matrix is to localize the memory usage, and it results in a more effective use of the cache memory.

The algorithm obtained by combining all the techniques described in this section is named MCS [25], [32], [34].

## 4. Improved Algorithms

### 4.1 An Approximate Solution as an Initial Lower Bound

When the algorithm MCS was first proposed in Ref. [25], 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 [25] is a local search algorithm based on our previous work [28]. It finds a near-maximum clique in a very short time, and the result is used as an initial lower bound 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 the original MCS [25]. Then $No_{th} := |Q_{\max}| - |Q|$ becomes large and the bounding condition becomes more effective.

The final version of MCS presented in Refs. [32], [34] excluded the procedure (init-lb) for finding an approximately maximum clique. This is because it is important to examine the performance of the main body of MCS [32] itself independently of an approximation algorithm.

We have many approximation algorithms for finding a maximum clique [37], while finding a good approximate solution for the maximum clique problem is considered to be very hard as shown that the maximum clique problem is not polynomial-time approximable within $|V|^{1-\epsilon}$ for any $\epsilon$ unless NP = P [38]. 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 Ref. [25] and choose another approximation algorithm called *k-opt local search* (KLS) by Katayama et al. [11]. It does not necessarily give the best quality solution, but it runs in quite a short time and it is easy to control the above trade-off. Note that KLS uses a random number in it for selecting a vertex. The KLS 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. This is because a vertex with a large degree tends to be included in a maximum clique. 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 choosed the number $rep$ of *repetitions* as follows by preliminary experiments in Ref. [35] with $d_0 = 1$:

$$rep = \min\{20n^{1/2} \times (\min\{dens, d_0\})^3,\ n\} \qquad \text{for } n \geq 1.$$

A procedure for finding an approximate maximum clique of the given graph $G = (V, E)$ under the above condition was named KLS($V, Q'_{\max}$) and its solution was given to $Q'_{\max}$ in Ref. [35]. The new MCS that was composed of a combination of the KLS procedure and MCS in Ref. [32] as above was named $\text{MCS}_1$.

In this paper, we take a slightly different approach from Ref. [35] by *giving more thought to problems requiring more than 1 second to solve by MCS*. Based on this intention, we try to improve the quality of the solution of KLS with a little more overhead. More exactly, we execute KLS just described above "5 times" with different random numbers (in parallel, conceptually) to get 5 (possibly different) solutions, but with $d_0 = 0.9$. And the final solution is set to be the best (largest) one among these 5 solutions. The total time required by the slightly extended KLS as above is about 5 times more than the time required by single execution of KLS in Ref. [35] but with possibly a better solution. Such a variant of KLS is named KLS5. We use in this paper KLS5($V, Q'_{\max}$) instead of KLS($V, Q'_{\max}$) in Ref. [35]. The KLS5($V, Q'_{\max}$) gives the (best of the 5) solution to $Q'_{\max}$. The new MCS that is composed of a combination of the KLS5 procedure and MCS in Ref. [32] as above is named KLS5_$\text{MCS}_1$ ($_{k5}\text{MCS}_1$ for short).

Recently, Batsyn et al. [2] and Maslov et al. [17] also demonstrated the effectiveness of an approximate solution, independently. They used iterated local search (ILS) heuristic developed by Andrade [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 ascending order with respect to their degrees prior to application of a branch-and-bound depth-first search for finding a maximum clique [6], [7], [27]. All of the preceding algorithms MCQ, MCR and MCS employ such sorting of vertices at the root level ($depth = 0$) of the search trees. Also clarified here is 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 [12].

Therefore, it becomes important to choose a good trade-off between the reduction of the search space and the time to achieve it. In order to solve this trade-off, we confirmed in Refs. [12], [19] and [23] that it is effective to adaptively control the search method. For an earlier algorithm MCLIQ [27] that is a predecessor of MCQ, we proposed a technique to solve the trade-off and reduced the overall running time successfully as follows [12]:

(i) At the first stage near the root of the search tree, we apply sorting of vertices followed by *Numbering*. (Ref. [12])

(ii) In the second stage of the search tree, we apply *Numbering* without new sorting of vertices. (Just as in Ref. [27])

(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 Ref. [7])

The above techniques are promising for any algorithm for finding a maximum clique if we control these three stages appropriately. We apply the techniques of Ref. [12] to MCS. Here, we make full use of the adjunct ordered set $V_a$ of vertices in MCS [32] in which vertices are sorted in ascending order with respect to their degrees from the rightmost (end) to the leftmost (front) by EXTENDED INITIAL SORT-NUMBER in Ref. [32]. In addition, we avoid the set $R$ of vertices in MCS [32] in which vertices are sorted with respect to their *Number*s. So, we are free from having to reconstruct such an $R$. From now on, we rename $V_a$ as $R$, for simplicity. So, be careful that the set $R$ in this paper corresponds to $V_a$ in MCS [32], and not to $R$ in MCS [32].

Hereafter, the NUMBERing procedure combined with Re-NUMBER is named NUMBER-R and is shown in **Fig. 3**. This is exactly the first half of the **procedure** Re-NUMBER-SORT in Fig. 2 of MCS [32].

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

```
procedure NUMBER-R(R, No)
begin
  {NUMBER}
    maxno := 0;
    C_1 := ∅;
    for i := 1 to |R| do
      { Conventional greedy approximate coloring }
      p := R[i] ;
      k := 1;
      while C_k ∩ Γ(p) ≠ ∅
        do k := k + 1 od
      if k > maxno then
        maxno := k;
        C_maxno := ∅
      fi
      C_k := C_k ∪ {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 = ∅ then
          maxno := maxno − 1
        fi
      fi
      { - Re-NUMBER ends - }

    od
end { of NUMBER-R }
```

**Fig. 3** Procedure NUMBER-R.

"**if** $(k > No_{th})$ **and** $(k = maxno)$ **then**" by

"**if** $(k > No_{th})$ **then**"

and

"Re-NUMBER-R" by

"Re-NUMBER1"

in NUMBER-R$(R, No)$. Thus, the condition for applying Re-NUMBER is relaxed in **procedure** NUMBER-R+$(R, No)$.

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. 5 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 Section 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 $|\{v \in newR \mid No[v] > No_{th}\}|/|newR|$ becomes large, then much more preprocessing time may be required. In addition, when *dens* (density) of the graph becomes larger it generally requires more time for finding a maximum clique and then much

```
procedure NUMBER-RL(R, No, newNo)
begin
  No_th := |Q_max| − |Q|;
  for i := 1 to |R| do
    C_i := ∅;
  od
  maxno := 1;
  for i := 1 to |R| do
    if No[R[i]] ≤ No_th then
      k := No[R[i]];
      if k > maxno then maxno := k fi
      C_k := C_k ∪ {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 ∩ Γ(p) ≠ ∅
        do k := k + 1 od
      if k > maxno then
        maxno := k;
      fi
      C_k := C_k ∪ {p};
      newNo[p] := k;
      if (k > No_th) then
        Re-NUMBER1(p, k, No, C_1, C_2, ..., C_maxno) ;
        if C_maxno = ∅ then
          maxno := maxno − 1
        fi
      fi
    fi
  od
end { of NUMBER-RL}
```

**Fig. 4** Procedure NUMBER-RL.

larger number of preprocessing steps is needed. As a result, we consider the following value:

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

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. 5. The new procedure obtained from Fig. 5 by replacing "$Th_1 := 0.4$, $Th_2 := 0.1$" by "$Th_1 := 0.4$, $Th_2 := 0$" at {Switches} is named $_{k5}$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 [13] also independently improved MCQ [29] successfully in a similar way.

### 4.3 Lightened Numbering Mainly near the Leaves of the Search Tree

Mainly near the leaves of the search tree, the ratio $|\{v \in newR \mid No[v] > No_{th}\}|/|newR|$ 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 Ref. [7] 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*

```
procedure k5_MCT(G = (V, E))
begin
  global Th_1 := 0.4;  Th_2 := 0.1;    {Switches}
  global Q := ∅;
  global Q_max := ∅;
  global dens := 2|E|/|V|(|V| − 1);  {density}
  if dens ≤ 0.1 then
    k5_MCS_1(G = (V, E));
  else
    Apply EXTENDED INITIAL SORT-NUMBER to V;
      { Q_max can be updated.}
    Reconstruct the adjacency matrix as described in [32];
    KLS5(V, Q'_max);
    if Q_max < Q'_max then
      Q_max := Q'_max
    fi
    NUMBER-R+(V, No);
      stage := 1;
    EXPAND(V, No, stage);
  fi
  output Q_max   {Maximum clique}
end { of k5_MCT}
```

**Fig. 5** Procedure $_{k5}$MCT.

```
procedure EXPAND(R, No, stage)
begin
  for i := |R| downto 1 do
    p := R[i];
    if (stage = 1 and |Q| + max_{v∈R}{No[v]} > |Q_max|)
    or (stage ≠ 1 and |Q| + No[p] > |Q_max|) then
      Q := Q ∪ {p};
      newR := R ∩ Γ(p);   {preserving the order}
      if newR ≠ ∅ then
        No_th := |Q_max| − |Q|;
        T := |{v ∈ newR | No[v] > No_th}| / |newR| × dens;
        if stage = 1 and Th_1 ≤ 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 ≤ T then
          NUMBER-R(newR, newNo);
          newstage := 2;
        else
          NUMBER-RL(newR, No, newNo);
          newstage := 3;
        fi
        EXPAND(newR, newNo, newstage)
      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. 6** Procedure EXPAND.

**Table 1** Comparison of CPU times in MCS, $_{k5}$MCS$_1$, $_{k5}$MCS$_2$ and $_{k5}$MCT.

| Graph | | | | KLS5 | times [sec] | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | $n$ | $dens$ | $\omega$ | $sol$ | MCS | $_{k5}$MCS$_1$ | $_{k5}$MCS$_2$ | $_{k5}$MCT | (MCS/$_{k5}$MCT)$_t$ |
| brock400_1 | 400 | 0.75 | 27 | 25 | 288 | 260 | 182 | 116 | 2.49 |
| brock400_2 | 400 | 0.75 | 29 | 25 | 124 | 104 | 77 | 47 | 2.66 |
| brock400_3 | 400 | 0.75 | 31 | 25 | 195 | 132 | 92 | 59 | 3.33 |
| brock400_4 | 400 | 0.75 | 33 | 25 | 103 | 101 | 73 | 46 | 2.23 |
| brock800_1 | 800 | 0.65 | 23 | 21 | 4,122 | 4,080 | 2,922 | 1,944 | 2.12 |
| brock800_2 | 800 | 0.65 | 24 | 21 | 3,683 | 3,484 | 2,499 | 1,629 | 2.26 |
| brock800_3 | 800 | 0.65 | 25 | 21 | 2,540 | 2,372 | 1,682 | 1,104 | 2.30 |
| brock800_4 | 800 | 0.65 | 26 | 20 | 1,768 | 1,758 | 1,256 | 819 | 2.16 |
| C250.9 | 250 | 0.90 | 44 | 44 | 1,171 | 946 | 779 | 405 | 2.89 |
| gen200_p0.9_44 | 200 | 0.90 | 44 | 44 | 0.17 | 0.39 | 0.39 | 0.29 | 0.60 |
| gen200_p0.9_55 | 200 | 0.90 | 55 | 55 | 0.46 | 0.32 | 0.32 | 0.32 | 1.43 |
| gen400_p0.9_55 | 400 | 0.90 | 55 | 55 | 22,536 | 80.8 | 89.3 | 1.18 | 19,098 |
| gen400_p0.9_65 | 400 | 0.90 | 65 | 65 | 57,385 | 6.81 | 7.04 | 1.44 | 39,851 |
| gen400_p0.9_75 | 400 | 0.90 | 75 | 75 | 108,298 | 2.44 | 2.43 | 1.33 | 81,427 |
| p_hat300-3 | 300 | 0.74 | 36 | 36 | 0.99 | 0.68 | 0.68 | 0.52 | 1.90 |
| p_hat500-3 | 500 | 0.75 | 50 | 50 | 57 | 35 | 33 | 18 | 3.11 |
| p_hat700-2 | 700 | 0.50 | 44 | 44 | 2.16 | 1.46 | 1.44 | 1.12 | 1.93 |
| p_hat700-3 | 700 | 0.75 | 62 | 62 | 900 | 470 | 440 | 217 | 4.16 |
| p_hat1000-2 | 1000 | 0.49 | 46 | 46 | 85 | 49 | 47 | 29 | 2.90 |
| p_hat1000-3 | 1000 | 0.74 | 68 | 68 | 305,146 | 111,587 | 96,207 | 39,134 | 7.80 |
| p_hat1500-2 | 1500 | 0.51 | 65 | 65 | 6,299 | 3,039 | 2,833 | 1,559 | 4.04 |
| san1000 | 1000 | 0.50 | 15 | 15 | 1.02 | 0.25 | 0.23 | 0.24 | 4.25 |
| san200_0.9_2 | 200 | 0.90 | 60 | 60 | 0.16 | 0.32 | 0.32 | 0.32 | 0.50 |
| san400_0.7_1 | 400 | 0.70 | 40 | 40 | 0.26 | 0.29 | 0.28 | 0.28 | 0.92 |
| san400_0.7_3 | 400 | 0.70 | 22 | 18 | 0.67 | 0.51 | 0.54 | 0.41 | 1.62 |
| sanr200_0.9 | 200 | 0.90 | 42 | 42 | 15.3 | 9.7 | 9.9 | 4.9 | 3.10 |
| sanr400_0.7 | 400 | 0.70 | 21 | 21 | 77 | 77 | 58 | 41 | 1.89 |
| DSJC1000.5 | 1000 | 0.50 | 15 | 15 | 141 | 137 | 125 | 94 | 1.50 |
| keller5 | 776 | 0.75 | 27 | 27 | 82,422 | 81,631 | 36,548 | 10,008 | 8.24 |
| frb30-15-1 | 450 | 0.82 | 30 | 30 | 740 | 434 | 464 | 75 | 9.84 |
| frb30-15-2 | 450 | 0.82 | 30 | 30 | 1,048 | 692 | 773 | 122 | 8.55 |
| frb30-15-3 | 450 | 0.82 | 30 | 28 | 670 | 678 | 729 | 124 | 5.39 |
| frb30-15-4 | 450 | 0.82 | 30 | 30 | 2,248 | 1,086 | 1,201 | 223 | 10.1 |
| frb30-15-5 | 450 | 0.82 | 30 | 30 | 972 | 741 | 780 | 105 | 9.29 |
| r200.8 | 200 | 0.80 | 24-27 | 24-27 | 1.64 | 1.47 | 1.31 | 0.90 | 1.82 |
| r200.9 | 200 | 0.90 | 39-44 | 39-44 | 26.5 | 19.0 | 19.2 | 10.7 | 2.48 |
| r200.95 | 200 | 0.95 | 58-66 | 58-66 | 19.4 | 12.0 | 12.5 | 10.9 | 1.78 |
| r250.8 | 250 | 0.80 | 26-28 | 26-28 | 19.6 | 17.5 | 13.9 | 8.8 | 2.23 |
| r300.8 | 300 | 0.80 | 28-29 | 28-29 | 160 | 133 | 96 | 59 | 2.73 |
| r400.7 | 400 | 0.70 | 21-22 | 21-22 | 75 | 67 | 50 | 35 | 2.15 |
| r400.8 | 400 | 0.80 | 30-31 | 30-31 | 6,467 | 5,527 | 3,437 | 1,997 | 3.24 |
| r500.7 | 500 | 0.70 | 22-23 | 21-23 | 733 | 692 | 473 | 326 | 2.25 |
| r1000.5 | 1000 | 0.50 | 15-16 | 14-15 | 134 | 133 | 123 | 92 | 1.46 |
| r1000.6 | 1000 | 0.60 | 19-20 | 18-19 | 6,591 | 6,569 | 4,862 | 3,456 | 1.91 |

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. 4**.

From preliminary experiments, we have chosen to turn to the new $stage = 3$ if the previously given value $T = (|\{v \in newR \mid No[v] > No_{th}\}|/|newR|) \times dens$ is less than 0.1. Then we let $Th_2 := 0.1$ in Fig. 5. The **procedure** NUMBER-RL is weaker than the previous **procedure** NUMBER-R for obtaining a strong bounding condition but 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. Thus, if $dens \leq 0.1$ we choose simpler algorithm $_{k5}$**MCS**$_1$ without relying on techniques introduced after $_{k5}$**MCS**$_1$ in this paper.

The resulting algorithm obtained by taking the total techniques in Sections 4.1–4.3 to improve MCS [32] is named $_{k5}$MCT (The 'T' is for 'Total'.) and is shown in **Fig. 5** and **Fig. 6**.

## 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, where the underlining programs were slightly revised from those in Ref. [35]. 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* [10] for `r300.5`, `r400.5` and `r500.5` are 0.141, 0.900 and 3.442 seconds, respectively. All the results of the experiments are summarized in **Tables 1–4**. Each first column lists the benchmark graphs brock - keller5 in DIMACS [10], the frb family in BHOSLIB [3], and/or random graphs r$n$.$p$ with the number of vertices=$n$ and the edge probability=$p$. As for the CPU time, the averages are taken over 10 random graphs r$n$.$p$ where $p < 0.9$. For random

**Table 2** Comparison of branches in MCS, $_{k5}$MCS$_1$, $_{k5}$MCS$_2$ and $_{k5}$MCT.

| Graph | | | | KLS5 | branches [$\times 10^{-6}$] | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Name | $n$ | *dens* | $\omega$ | *sol* | MCS | $_{k5}$MCS$_1$ | $_{k5}$MCS$_2$ | $_{k5}$MCT | (MCS/$_{k5}$MCT)$_b$ |
| brock400_1 | 400 | 0.75 | 27 | 25 | 89 | 77 | 52 | 55 | 1.62 |
| brock400_2 | 400 | 0.75 | 29 | 25 | 34 | 24 | 18 | 18 | 1.82 |
| brock400_3 | 400 | 0.75 | 31 | 25 | 65 | 38 | 25 | 27 | 2.41 |
| brock400_4 | 400 | 0.75 | 33 | 25 | 31 | 30 | 20 | 22 | 1.42 |
| brock800_1 | 800 | 0.65 | 23 | 21 | 1,092 | 1,081 | 748 | 786 | 1.39 |
| brock800_2 | 800 | 0.65 | 24 | 21 | 953 | 867 | 599 | 627 | 1.52 |
| brock800_3 | 800 | 0.65 | 25 | 21 | 630 | 557 | 378 | 396 | 1.59 |
| brock800_4 | 800 | 0.65 | 26 | 20 | 381 | 380 | 258 | 270 | 1.41 |
| C250.9 | 250 | 0.90 | 44 | 44 | 255 | 197 | 154 | 188 | 1.35 |
| gen200_p0.9_44 | 200 | 0.90 | 44 | 44 | 0.0355 | 0.0170 | 0.0167 | 0.0024 | 14.5 |
| gen200_p0.9_55 | 200 | 0.90 | 55 | 55 | 0.11248 | 0.00073 | 0.00065 | 0.00060 | 188 |
| gen400_p0.9_55 | 400 | 0.90 | 55 | 55 | 2,895 | 7.4348 | 8.1264 | 0.0002 | 11,626,243 |
| gen400_p0.9_65 | 400 | 0.90 | 65 | 65 | 7,628 | 0.330 | 0.341 | 0.058 | 131,671 |
| gen400_p0.9_75 | 400 | 0.90 | 75 | 75 | 17,153 | 0.054 | 0.052 | 0.002 | 8,354,874 |
| p_hat300-3 | 300 | 0.74 | 36 | 36 | 0.23 | 0.07 | 0.07 | 0.09 | 2.63 |
| p_hat500-3 | 500 | 0.75 | 50 | 50 | 7.9 | 4.3 | 4.1 | 5.6 | 1.41 |
| p_hat700-2 | 700 | 0.50 | 44 | 44 | 0.339 | 0.126 | 0.121 | 0.197 | 1.72 |
| p_hat700-3 | 700 | 0.75 | 62 | 62 | 88 | 43 | 40 | 54 | 1.64 |
| p_hat1000-2 | 1000 | 0.49 | 46 | 46 | 12.6 | 6.6 | 6.3 | 10.0 | 1.26 |
| p_hat1000-3 | 1000 | 0.74 | 68 | 68 | 27,212 | 9,026 | 7,822 | 9,027 | 3.01 |
| p_hat1500-2 | 1500 | 0.51 | 65 | 65 | 560 | 253 | 234 | 400 | 1.40 |
| san1000 | 1000 | 0.50 | 15 | 15 | 0.085 | 0.0005 | 0 | 0 | |
| san200_0.9_2 | 200 | 0.90 | 60 | 60 | 0.042 | 0.0001 | 0 | 0 | |
| san400_0.7_1 | 400 | 0.70 | 40 | 40 | 0.023 | 0.0002 | 0 | 0 | |
| san400_0.7_3 | 400 | 0.70 | 22 | 18 | 0.124 | 0.041 | 0.050 | 0.054 | 2.32 |
| sanr200_0.9 | 200 | 0.90 | 42 | 42 | 3.5 | 2.0 | 1.9 | 2.1 | 1.63 |
| sanr400_0.7 | 400 | 0.70 | 21 | 21 | 30 | 29 | 21 | 23 | 1.31 |
| DSJC1000.5 | 1000 | 0.50 | 15 | 15 | 52 | 49 | 43 | 45 | 1.14 |
| keller5 | 776 | 0.75 | 27 | 27 | 13,148 | 13,152 | 5,519 | 4,495 | 2.93 |
| frb30-15-1 | 450 | 0.82 | 30 | 30 | 157 | 82 | 85 | 38 | 4.12 |
| frb30-15-2 | 450 | 0.82 | 30 | 30 | 229 | 135 | 148 | 65 | 3.55 |
| frb30-15-3 | 450 | 0.82 | 30 | 28 | 147 | 146 | 153 | 73 | 2.01 |
| frb30-15-4 | 450 | 0.82 | 30 | 30 | 509 | 214 | 232 | 121 | 4.22 |
| frb30-15-5 | 450 | 0.82 | 30 | 30 | 203 | 143 | 146 | 57 | 3.58 |
| r200.8 | 200 | 0.80 | 24-27 | 24-27 | 0.66 | 0.51 | 0.42 | 0.47 | 1.40 |
| r200.9 | 200 | 0.90 | 39-44 | 39-44 | 6.5 | 4.4 | 4.1 | 5.2 | 1.25 |
| r200.95 | 200 | 0.95 | 58-66 | 58-66 | 2.7 | 1.5 | 1.5 | 2.5 | 1.09 |
| r250.8 | 250 | 0.80 | 26-28 | 26-28 | 7.0 | 6.0 | 4.5 | 4.9 | 1.42 |
| r300.8 | 300 | 0.80 | 28-29 | 28-29 | 53 | 42 | 29 | 31 | 1.69 |
| r400.7 | 400 | 0.70 | 21-22 | 21-22 | 28 | 24 | 17 | 18 | 1.54 |
| r400.8 | 400 | 0.80 | 30-31 | 30-31 | 1,921 | 1,588 | 933 | 999 | 1.92 |
| r500.7 | 500 | 0.70 | 22-23 | 21-23 | 261 | 241 | 159 | 170 | 1.54 |
| r1000.5 | 1000 | 0.50 | 15-16 | 14-15 | 50 | 49 | 43 | 45 | 1.11 |
| r1000.6 | 1000 | 0.60 | 19-20 | 18-19 | 2,137 | 2,111 | 1,513 | 1,594 | 1.34 |

graphs r$n.p$ where $p \geq 0.9$, the averages are taken over 100 random graphs. The random graphs in this paper are not exactly the same as those in Ref. [35]. In the Tables, $n$ and $\omega$ stands for the number of vertices and the size of the maximum clique, respectively. In addition, *dens* stand for *density* of a benchmark graph and $p$ for a random graph r$n.p$. The columns *sol* and *time* below KLS5 show the solution and the computing time of KLS5, respectively.

### 5.1 Stepwise Improvement

Tables 1 and 2 show stepwise improvement from MCS to $_{k5}$**MCT** for selected graphs chosen from the following Tables 3 and 4. In Table 1, (MCS/$_{k5}$MCT)$_t$ is the ratio of the CPU time required by MCS to that of $_{k5}$MCT. In Table 2, (MCS/$_{k5}$MCT)$_b$ is the ratio of the number of branches required by MCS to that of $_{k5}$MCT.

**(1)** Improvement from MCS to $_{k5}$**MCS**$_1$ by an approximate solution given by KLS5 in Section 4.1: The improvement is particularly quite effective for the gen graph family. $_{k5}$MCS$_1$ is faster than MCS for gen400_p0.9_75, gen400_p0.9_65, and gen400_p0.9_55 by more than 44,000, 8,000, and 270 times, respectively. This technique is effective for almost all graphs but with few exceptions such as the MANN graph family.

**(2)** Improvement from $_{k5}$MCS$_1$ to $_{k5}$**MCS**$_2$ by EXTENDED INITIAL SORT-NUMBER in Section 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 $_{k5}$MCS$_2$ to $_{k5}$**MCT** by Lightened Numbering in Section 4.3: This technique is effective for almost all graphs in reducing computing time in spite of increased numbers of branches in general. The $_{k5}$MCT is faster than $_{k5}$MCS$_2$ for gen400_p0.9_55, frb30-15-5, frb30-15-2, and frb30-15-1 by more than 76, 7, 6, and 6 times, respectively, where their numbers of branches are also reduced.

### 5.2 Overall Improvement

Table 3 shows the result of the overall improvement from MCS to $_{k5}$MCT in computing time for the benchmark graphs. The column *time* below KLS5 shows the computing time for KLS5 that

**Table 3** CPU time [sec] for benchmark graphs.

| Graph | | | | KLS5 | | MCS | $_{k5}$MCT | BBMCX | MaxCLQ | ILS&MCS | BG14 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | $n$ | *dens* | $\omega$ | *sol* | *time* | [32] | | [22] | [15] | [17] | [2] |
| brock200_1 | 200 | 0.75 | 21 | 21 | 0.09 | 0.36 | 0.30 | **0.18** | 0.34 | 4.42 | 2.51 |
| brock400_1 | 400 | 0.75 | 27 | 25 | 0.41 | 288 | **116** | 150 | 205 | 188 | 302 |
| brock400_2 | 400 | 0.75 | 29 | 25 | 0.41 | 124 | **47** | 68 | 96 | 94 | 132 |
| brock400_3 | 400 | 0.75 | 31 | 25 | 0.41 | 195 | **59** | 120 | 160 | 145 | 211 |
| brock400_4 | 400 | 0.75 | 33 | 25 | 0.41 | 103 | **46** | 68 | 100 | 72 | 87 |
| brock800_1 | 800 | 0.65 | 23 | 21 | 1.07 | 4,122 | **1,944** | 2,690 | 4,562 | 3,998 | 4,216 |
| brock800_2 | 800 | 0.65 | 24 | 21 | 1.09 | 3,683 | **1,629** | 2,415 | 4,002 | 3,462 | 3,778 |
| brock800_3 | 800 | 0.65 | 25 | 21 | 1.07 | 2,540 | **1,104** | 1,587 | 2,510 | 2,361 | 2,649 |
| brock800_4 | 800 | 0.65 | 26 | 20 | 1.08 | 1,768 | **819** | 1,100 | 1,853 | 1,685 | 1,868 |
| C250.9 | 250 | 0.90 | 44 | 44 | 0.43 | 1,171 | 405 | 713 | **268** | | |
| C2000.5 | 2000 | 0.50 | 16 | 16 | 2.74 | 33,899 | **21,030** | | | | |
| gen200_p0.9_44 | 200 | 0.90 | 44 | 44 | 0.29 | 0.17 | 0.29 | 0.16 | **0.11** | 1.68 | |
| gen200_p0.9_55 | 200 | 0.90 | 55 | 55 | 0.32 | 0.46 | 0.32 | 0.31 | **0.14** | 2.43 | 0.92 |
| gen400_p0.9_55 | 400 | 0.90 | 55 | 55 | 1.18 | 22,536 | **1.18** | 19,362 | | 46,504 | 2,965 |
| gen400_p0.9_65 | 400 | 0.90 | 65 | 65 | 1.23 | 57,385 | **1.44** | 66,135 | 36,684 | 2,130 | 18 |
| gen400_p0.9_75 | 400 | 0.90 | 75 | 75 | 1.32 | 108,298 | **1.33** | 47,176 | 9,984 | 84 | 7.8 |
| MANN_a27 | 378 | 0.99 | 126 | 126 | 2.60 | 0.26 | 2.88 | 0.17 | **0.16** | 1.30 | |
| MANN_a45 | 1035 | 0.99 | 345 | 344 | 79 | 53 | 128 | 32 | 23 | **17** | 55 |
| p_hat300-3 | 300 | 0.74 | 36 | 36 | 0.31 | 0.99 | **0.52** | 0.66 | 1.16 | 6.72 | 3.62 |
| p_hat500-3 | 500 | 0.75 | 50 | 50 | 1.11 | 57.1 | **18.4** | 33.3 | 39.6 | 50 | 60 |
| p_hat700-2 | 700 | 0.50 | 44 | 44 | 0.57 | 2.16 | **1.12** | 1.53 | 3.61 | 59 | 29 |
| p_hat700-3 | 700 | 0.75 | 62 | 62 | 2.28 | 900 | **217** | 680 | 879 | 552 | 767 |
| p_hat1000-1 | 1000 | 0.25 | 10 | 10 | 0.02 | 0.23 | 0.22 | **0.19** | 1.60 | 218 | |
| p_hat1000-2 | 1000 | 0.49 | 46 | 46 | 1.13 | 85 | **29** | 73 | 101 | 204 | 113 |
| p_hat1000-3 | 1000 | 0.74 | 68 | 68 | 4.94 | 305,146 | **39,134** | | | | |
| p_hat1500-1 | 1500 | 0.25 | 12 | 11 | 0.08 | 1.82 | **1.46** | 1.95 | 10 | 478 | 422 |
| p_hat1500-2 | 1500 | 0.51 | 65 | 65 | 4.77 | 6,299 | **1,559** | 3,852 | 8,027 | 5,346 | 5,434 |
| san1000 | 1000 | 0.50 | 15 | 15 | 0.23 | 1.02 | **0.24** | 0.68 | 0.72 | 449 | 158 |
| san200_0.9_2 | 200 | 0.90 | 60 | 60 | 0.32 | 0.16 | 0.32 | **0.07** | 0.10 | 12 | |
| san400_0.7_1 | 400 | 0.70 | 40 | 40 | 0.28 | 0.26 | 0.28 | 0.14 | **0.13** | 16 | 7 |
| san400_0.7_2 | 400 | 0.70 | 30 | 30 | 0.25 | **0.059** | 0.256 | 0.092 | 0.064 | 19 | |
| san400_0.7_3 | 400 | 0.70 | 22 | 18 | 0.25 | 0.67 | 0.41 | **0.39** | 0.43 | 27 | 12 |
| sanr200_0.7 | 200 | 0.70 | 18 | 18 | 0.07 | 0.15 | 0.16 | **0.08** | 0.17 | 5.05 | 1.03 |
| sanr200_0.9 | 200 | 0.90 | 42 | 42 | 0.30 | 15.33 | 4.94 | 7.38 | **4.21** | 4.62 | 10.2 |
| sanr400_0.5 | 400 | 0.50 | 13 | 13 | 0.07 | 0.35 | 0.31 | **0.19** | 0.69 | 35 | 18 |
| sanr400_0.7 | 400 | 0.70 | 21 | 21 | 0.31 | 77 | **41** | 44 | 81 | 86 | 81 |
| DSJC500.5 | 500 | 0.50 | 13 | 13 | 0.12 | 1.5 | 1.3 | **0.8** | 2.8 | | |
| DSJC1000.5 | 1000 | 0.50 | 15 | 15 | 0.57 | 141 | **94** | 102 | 265 | | |
| keller5 | 776 | 0.75 | 27 | 27 | 1.65 | 82,422 | 10,008 | 30,299 | **4,982** | 5,777 | 82,508 |
| frb30-15-1 | 450 | 0.82 | 30 | 30 | 0.76 | 740 | **75** | 1,029 | 560 | | |
| frb30-15-2 | 450 | 0.82 | 30 | 30 | 0.75 | 1,048 | **122** | 672 | 758 | | |
| frb30-15-3 | 450 | 0.82 | 30 | 28 | 0.75 | 670 | **124** | 350 | 477 | | |
| frb30-15-4 | 450 | 0.82 | 30 | 30 | 0.75 | 2,248 | **223** | 1,157 | 955 | | |
| frb30-15-5 | 450 | 0.82 | 30 | 30 | 0.75 | 972 | **105** | 801 | 705 | | |

is a part of the total CPU time. In Table 2 of Ref. [35], the *time* included the computing time from "Apply *EXTENDED INITIAL SORT-NUMBER* to $V$;" to "Reconstruct the adjacency matrix as described in Ref. [32]," since sorting of vertices is necessary before KLS5($V, Q'_{\max}$). But, the *time* in this paper excludes the above preceding time so that KLS5 itself is clearer. Table 4 shows the same result for random graphs.

Tables 3–4 include the state-of-the-art result of BBMCX [22] by Segundo et al. that makes good use of a limited MaxSAT bound. Here, its computing time is calibrated in the established way in the Second DIMACS Implementation Challenge [10], where our computer is calculated to be 1.30 times faster than that in Ref. [22]. (See **Table 5** in Appendix for the detail, and that $1.30 > 1.298$.) From Ref. [22], Tables 3–4 also include the calibrated computing time of MaxCLQ [14], [15] by Li and Quan that is based on MaxSAT. The calibrated computing time by ILS&MCS [17] and BG14 [2] are added on the assumption that the performance of each MCS is the same as that in this paper, for reference, too. The boldface entries indicate the fastest time in the row.

The result shows that the $_{k5}$MCT is faster than MCS for gen400_p0.9_75, gen400_p0.9_65 and gen400_p0.9_55 by over 81,000, 39,000 and 19,000 times, respectively. The $_{k5}$MCT is faster than MCS for frb30-15-4 by over 10 times. The $_{k5}$MCT is faster than MCS for frb30-15-1, frb30-15-5, frb30-15-2, keller5, p_hat1000-3, frb30-15-3 by over 5 times. The $_{k5}$MCT is faster than MCS for san1000, p_hat700-3, p_hat1500-2, brock400_3, r400.8, p_hat500-3, sanr200_0.9 by over 3 times. In Table 1, $_{k5}$MCT is faster than MCS by more than 2 times for the other 14 graphs including r300.8, r200.9, r500.7, r250.8 and r400.7.

Except for few special graphs such as in the MANN family and for easy graphs that can be solved in less than 1 second by MCS, $_{k5}$MCT is faster than or equal to MCS for all graphs in Tables 3 and 4.

The $_{k5}$MCT is faster than the other algorithms in Tables 3 and 4 for many instances. Note that MaxCLQ is fast for dense graphs. ILS&MCS [17] and BG14 [2] require more time than $_{k5}$MCT for most of the instances tested. One reason for this difference comes from the fact that our approximation algorithm, KLS, takes up only small portion of the total algorithm's computing time with

**Table 4** CPU time [sec] for random graphs.

| Graph | | | | KLS5 | | MCS | $_{k5}$MCT | BBMCX | MaxCLQ |
|---|---|---|---|---|---|---|---|---|---|
| Name | $n$ | $dens$ | $\omega$ | $sol$ | $time$ | [32] | | [22] | [15] |
| r150.9 | 150 | 0.90 | 35-39 | 35-39 | 0.14 | 0.40 | 0.30 | 0.26 | **0.13** |
| r150.95 | 150 | 0.95 | 52-57 | 52-57 | 0.24 | 0.13 | 0.28 | 0.11 | **0.02** |
| r200.7 | 200 | 0.70 | 17-19 | 17-19 | 0.07 | 0.17 | 0.17 | **0.09** | 0.18 |
| r200.8 | 200 | 0.80 | 24-27 | 24-27 | 0.13 | 1.64 | **0.90** | 0.95 | 1.08 |
| r200.9 | 200 | 0.90 | 39-44 | 39-44 | 0.29 | 26.5 | 10.7 | 14.8 | **6.2** |
| r200.95 | 200 | 0.95 | 58-66 | 58-66 | 0.49 | 19.4 | 10.9 | 30.2 | **2.5** |
| r250.7 | 250 | 0.70 | 18-19 | 18-19 | 0.11 | 1.07 | **0.75** | | |
| r250.8 | 250 | 0.80 | 26-28 | 26-28 | 0.20 | 19.6 | **8.8** | | |
| r300.6 | 300 | 0.60 | 15-16 | 15-16 | 0.09 | 0.47 | 0.41 | **0.21** | 0.58 |
| r300.7 | 300 | 0.70 | 20-20 | 19-20 | 0.16 | 5.3 | 2.9 | **2.6** | 4.7 |
| r300.8 | 300 | 0.80 | 28-29 | 28-29 | 0.30 | 160 | **59** | 89 | 87 |
| r400.5 | 400 | 0.50 | 13-13 | 12-13 | 0.07 | 0.32 | **0.30** | | |
| r400.6 | 400 | 0.60 | 16-17 | 16-17 | 0.16 | 3.59 | **2.41** | | |
| r400.7 | 400 | 0.70 | 21-22 | 21-22 | 0.31 | 75 | **35** | | |
| r400.8 | 400 | 0.80 | 30-31 | 30-31 | 0.55 | 6,467 | **1,997** | | |
| r500.5 | 500 | 0.50 | 13-14 | 13-13 | 0.11 | 1.31 | 1.17 | **0.64** | 2.09 |
| r500.6 | 500 | 0.60 | 17-18 | 16-17 | 0.26 | 18.0 | 11.3 | **10.1** | 22.1 |
| r500.7 | 500 | 0.70 | 22-23 | 21-23 | 0.49 | 733 | **326** | 423 | 564 |
| r600.4 | 600 | 0.40 | 11-11 | 10-11 | 0.05 | 0.36 | **0.35** | | |
| r600.5 | 600 | 0.50 | 14-14 | 13-14 | 0.17 | 4.21 | **3.29** | | |
| r600.6 | 600 | 0.60 | 17-18 | 17-18 | 0.40 | 85 | **48** | | |
| r1000.3 | 1000 | 0.30 | 9-10 | 8-9 | 0.03 | 0.46 | 0.46 | **0.38** | 2.03 |
| r1000.4 | 1000 | 0.40 | 12-12 | 11-11 | 0.17 | 5.9 | 5.3 | **4.5** | 14.5 |
| r1000.5 | 1000 | 0.50 | 15-16 | 14-15 | 0.54 | 134 | **92** | 103 | 231 |
| r1000.6 | 1000 | 0.60 | 19-20 | 18-19 | 1.25 | 6,591 | **3,456** | | |
| r2000.2 | 2000 | 0.20 | 8-8 | 7-8 | 0.02 | **0.67** | 0.69 | | |
| r2000.3 | 2000 | 0.30 | 10-11 | 9-10 | 0.17 | 15.0 | **13.5** | | |
| r2000.4 | 2000 | 0.40 | 13-14 | 12-13 | 0.83 | 452 | **362** | | |
| r3000.1 | 3000 | 0.10 | 6-7 | 5-6 | 0.00 | 0.21 | 0.21 | **0.19** | 15 |
| r3000.2 | 3000 | 0.20 | 9-9 | 7-8 | 0.05 | 3.62 | **3.48** | 4.34 | 34 |
| r3000.3 | 3000 | 0.30 | 11-11 | 10-10 | 0.44 | 121 | **108** | | |
| r3000.4 | 3000 | 0.400 | 14-14 | 12-13 | 2.28 | 6,411 | **5,176** | | |
| r4000.1 | 4000 | 0.10 | 7-7 | 5-7 | 0.00 | **0.53** | **0.53** | | |
| r4000.2 | 4000 | 0.20 | 9-9 | 8-9 | 0.10 | 14.8 | **13.4** | | |
| r4000.3 | 4000 | 0.30 | 11-12 | 10-11 | 0.88 | 633 | **547** | | |
| r5000.1 | 5000 | 0.10 | 7-7 | 5-6 | 0.01 | **1.13** | **1.13** | 1.19 | 68 |
| r5000.2 | 5000 | 0.20 | 9-10 | 8-8 | 0.20 | 44 | **39** | 68 | 578 |
| r5000.3 | 5000 | 0.30 | 12-12 | 10-11 | 1.55 | 2,269 | **1,873** | | |
| r10000.05 | 10000 | 0.05 | 6-6 | 4-5 | 0.01 | **1.883** | 1.886 | | |
| r10000.1 | 10000 | 0.10 | 7-7 | 6-7 | 0.04 | 13.7 | **13.1** | 20 | 684 |
| r10000.2 | 10000 | 0.20 | 10-10 | 8-9 | 0.87 | 1,331 | **1,136** | | |
| r15000.05 | 15000 | 0.05 | 6-6 | 5-5 | 0.02 | **6.96** | **6.96** | | |
| r15000.1 | 15000 | 0.10 | 8-8 | 6-6 | 0.10 | 65.3 | **61.3** | 115 | 2,749 |
| r15000.2 | 15000 | 0.20 | 10-11 | 9-9 | 2.18 | 10,565 | **9,488** | | |
| r20000.05 | 20000 | 0.05 | 6-7 | 5-5 | 0.05 | **18.3** | **18.3** | | |
| r20000.1 | 20000 | 0.10 | 8-8 | 6-7 | 0.17 | 251 | **235** | | |

few exceptions, whereas their approximation algorithm, ILS [1] in ILS&MCS and BG14, consumes a considerable part of the total computing time. To be more precise, they run the ILS heuristic with 100,000 scans for all the considered instances except gen400_p0.9_55 and p_hat1000-3 for which they use 60 million scans because these two instances are computationally difficult. Their resulting approximate solution for gen400_p0.9_55 is 54, whereas our corresponding solution is 55 that is optimal.

In $_{k5}$MCT, even if we change the value of $d_0$ from 0.9 to 1 in $rep$ we have no change of $sol$ in Tables 1–4. So, reducing $d_0 = 1$ in MCT [35] to $d_0 = 0.9$ in $_{k5}$MCT is reasonable.

## 6. Concluding Remarks

In conclusion, $_{k5}$MCT has achieved significant improvement over MCS, that is, $_{k5}$MCT is much faster than MCS for graphs which require more than around 1 second to solve by MCS [32].

For comparison, the number of graphs for which $_{k5}$MCT is faster than MCS by more than 1000, 100, 10, and 5 times is 3, 3, 4, and 10, respectively, in Table 1, whereas the number of graphs for which MCT is faster than MCS by more than 1000, 100, 10, and 5 times is 2, 3, 3, and 9, respectively, among the same group of graphs [35]. This is an example to show that $_{k5}$MCT is slightly faster than MCT for graphs which require more than around 1 second to solve by MCS, but with few exceptions as in MANN family graphs. See Ref. [35] for the details of MCT.

Another improved MCT could be obtained by replacing simply

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

in Ref. [35] by

$$rep = 5 \times \min\{20n^{1/2} \times dens^3,\ n\} \quad \text{for } n \geq 1,$$

and we call it $_{r5}$MCT, where the modified KLS in this way is named KLSr5. This KLSr5 takes almost the same time as KLS5 for the same instance. As one example for gen400_p0.9_55, $_{r5}$MCT gets an approximately maximum clique of size 53 and requires the total of 135.9 seconds to obtain the final exact result

of 55, whereas $_{k5}$MCT gets an approximately maximum clique of size 55 (= exact solution) and requires a total of 1.18 seconds to obtain the final result as shown in Table 1. If we let $rep := 23{,}852$ in this $_{r5}$MCT then it just manages to get an approximately maximum clique of size 55 and requires the total of 19.16 seconds to obtain the final exact result. This shows another example of the advantage of $_{k5}$MCT over MCT, hence also over MCS.

It is left as an important problem to choose other better approximation algorithms for the maximum clique problem. The present algorithm can be easily extended for enumerating all maximal cliques of the maximum and near-maximum size with the technique of Ref. [30].

**Acknowledgments** We are thankful to K. Yoshida and T. Hatta for their contribution at the preceding stage of this work. We express our sincere gratitude to K. Katayama, E. Harley, T. Toda, and T. Nishino for their useful comments and help. This work was supported in part by JSPS KAKENHI Grant Numbers JP22500009, JP24650006, JP2530009, JP15K11985, 17K00006, MEXT KAKENHI Grant Number JP24106003, JST CREST grant JPMJCR1402, and Kayamori Foundation of Informational Science Advancement.

## References

[1] Andrade, D.V., Resende, M.G.C. and Werneck, R.F.: Fast local search for the maximum independent set problem, *J. Heuristics*, Vol.18, pp.525–547 (2012).

[2] Batsyn, M., Goldengorin, B., Maslov, E. and Pardalos, P.M.: Improvements to MCS algorithm for the maximum clique problem, *J. Comb. Optim.*, Vol.27, pp.397–416 (2014).

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

[4] Bomze, I.M., Budinich, M., Pardalos, P.M. and Pelillo M.: The Maximum Clique Problem, Du, D.-Z. and Pardalos, P.M. (Eds.), *Handbook of Combinatorial Optimization*, Supplement Vol.A, pp.1–74, Kluwer Academic Publishers (1999).

[5] Butenko, S. and Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics - Invited Review, *European J. Operational Research*, Vol.173, pp.1–17 (2006).

[6] Carraghan, R. and Pardalos, P.M.: An exact algorithm for the maximum clique problem, *Operations Research Letters*, Vol.9, pp.375–382 (1990).

[7] Fujii, T. and Tomita, E.: On efficient algorithms for finding a maximum clique, Technical Report of IECE, AL81-113, pp.25–34 (1982).

[8] Hatta, T., Tomita, E., Ito, H. and Wakatsuki, M.: An improved branch-and-bound algorithm for finding a maximum clique, *Proc. Summer LA Symposium*, No.9, pp.1–8 (2015).

[9] Hotta, K., Tomita, E. and Takahashi, H.: A view-invariant human face detection method based on maximum cliques, *Trans. IPSJ*, Vol.44, SIG14 (TOM9), pp.57–70 (2003).

[10] Johnson, D.S. and Trick, M.A. (Eds.): *Cliques, Coloring, and Satisfiability, DIMACS Series in DMTCS*, Vol.26, American Math. Soc. (1996).

[11] Katayama, K., Hamamoto, A. and Narihisa, H.: An effective local search for the maximum clique problem, *Inf. Process. Lett.*, Vol.95, pp.503–511 (2005).

[12] Kohata, Y., Nishijima, T., Tomita, E., Fujihashi, C. and Takahashi, H.: Efficient algorithms for finding a maximum clique, Technical Report of IEICE, COMP89-113, pp.1–8 (1990).

[13] Konc, J. and Janežič, D.: An improved branch and bound algorithm for the maximum clique problem, *MATCH Commun. Math. Comput. Chem.*, Vol.58, pp.569–590 (2007).

[14] Li, C.M. and Quan, Z.: An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem, *AAAI Conf. AI*, pp.128–133 (2010).

[15] Li, C.M. and Quan, Z.: Combining graph structure exploitation and propositional reasoning for the maximum clique problem, *Proc. IEEE ICTAI*, pp.344–351 (2010).

[16] Liu, E., Zhang, Q. and Leung, K.K.: Clique-based utility maximization in wireless mesh networks, *IEEE Trans. Wireless Commun.*, Vol.10, pp.948–957 (2011).

[17] Maslov, E., Batsyn, M. and Pardalos, P.M.: Speeding up branch and bound algorithms for solving the maximum clique problem, *J. Global Optim.*, Vol.59, pp.1–21 (2014).

[18] Mori, T., Tamura, T., Fukagawa, D., Takasu, A., Tomita, E. and Akutsu, T.: A clique-based method using dynamic programming for computing edit distance between unordered trees, *Journal of Computational Biology*, Vol.19, pp.1089–1104 (2012).

[19] Nagai, M., Tabuchi, T., Tomita, E. and Takahashi, H.: An experimental evaluation of some algorithms for finding a maximum clique, *Conf. Records of the National Convention of IEICE 1988*, D-348 (1988).

[20] Ogawa, H.: Labeled point pattern matching by Delaunay triangulation and maximal cliques, *Pattern Recognition*, Vol.19, pp.35–40 (1986).

[21] Pardalos, P.M. and Xue, J.: The maximum clique problem, *J. Global Optim.*, Vol.4, pp.301–328 (1994).

[22] Segundo, P.S., Nikolaev, A. and Batsyn, M.: Infra-chromatic bound for exact maximum clique search, *Computers and Operations Research*, Vol.64, pp.293–303 (2015).

[23] Shindo, M., Tomita, E. and Maruyama, Y.: An efficient algorithm for finding a maximum clique, Technical Report of IECE, CAS86-5, pp.33–40 (1986).

[24] Sloan, N.J.A.: Challenge Problems: Independent Sets in Graphs, available from ⟨https://oeis.org/A265032/a265032.html⟩.

[25] Sutani, Y., Higashi, T., Tomita, E., Takahashi, S. and Nakatani, H.: A faster branch-and-bound algorithm for finding a maximum clique, Technical Report of IPSJ, 2006-AL-108, pp.79–86 (2006).

[26] Tomita, E. and Yamada, M.: An algorithm for finding a maximum complete subgraph, *Conf. Records of the National Convention of IECE 1978*, p.8 (1978).

[27] Tomita, E., Kohata, Y. and Takahashi, H.: A simple algorithm for finding a maximum clique, Technical Report of the University of Electro-Communications, UEC-TR-C5(1) (1988). (Reference [239] in Ref. [21] and Reference [308] in Ref. [4]), available from ⟨http://id.nii.ac.jp/1438/00001899/⟩.

[28] Tomita, E., Mitsuma, S. and Takahashi, H.: Two algorithms for finding a near-maximum clique, Technical Report of the University of Electro-Communications, UEC-TR-C1 (1988). (Reference [240] in Ref. [21] and Reference [309] in Ref. [4]), available from ⟨http://id.nii.ac.jp/1438/00001900/⟩.

[29] Tomita, E. and Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique, *DMTCS 2003*, *LNCS 2731*, pp.278–289 (2003).

[30] Tomita, E., Tanaka, A. and Takahashi, H. : The worst-case time complexity for generating all maximal cliques and computational experiments, *Theoret. Comput. Sci.*, Vol.363 (Special Issue on COCOON 2004), pp.28–42 (2006).

[31] Tomita, E. and Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, *J. Global Optim.*, Vol.37, pp.95–111 (2007), *J. Global Optim.*, Vol.44, p.311 (2009).

[32] Tomita, E., Sutani, Y., Higashi, T., Takahashi, S. and Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique, *WALCOM 2010*, *LNCS 5942*, pp.191–203 (2010).

[33] Tomita, E., Akutsu, T. and Matsunaga, T.: Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics, Laskovski, A.N. (Ed.): *Biomedical Engineering, Trends in Electronics, Communications and Software*, pp.625–640, InTech (2011).

[34] Tomita, E., Sutani, Y., Higashi, T. and Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments, *IEICE Trans. Inf. Syst.*, Vol.E96-D, pp.1286–1298 (2013), available from ⟨http://id.nii.ac.jp/1438/00000287/⟩.

[35] Tomita, E., Yoshida, K., Hatta, T., Nagao, A., Ito, H. and Wakatsuki, M.: A much faster branch-and-bound algorithm for finding a maximum clique, *FAW 2016*, *LNCS 9711*, pp.215–226 (2016).

[36] Tomita, E.: Efficient algorithms for finding maximum and maximal cliques and their applications, *Keynote at WALCOM 2017*, *LNCS 10167*, pp.3–15 (2017).

[37] Wu, Q. and Hao, J.K.: A review on algorithms for maximum clique problems – Invited Review, *European J. Operational Research*, Vol.242, pp.693–709 (2015).

[38] Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number, *Proc. STOC 2006*, pp.681–690 (2006).

## Appendix: Clique benchmark result

**Table 5** Each dfmax running time for instances [sec].

| | $_{k5}$.MCT This paper | | BBMCX Segundo et al. [22] |
|---|---|---|---|
| Graph | $T_1$ | $T_2$ | $T_2/T_1$ |
| r300.5 | 0.141 | 0.189 | 1.340 |
| r400.5 | 0.900 | 1.155 | 1.283 |
| r500.5 | 3.442 | 4.369 | 1.269 |
| | | | Average: 1.298 |

**Etsuji Tomita** received his B.Eng. and Dr.Eng. degrees in Electronics Engineering from Tokyo Institute of Technology (Tokyo Tech), in 1966 and 1971, respectively. Then he was with the faculties of Tokyo Tech, and was appointed an Associate Professor and subsequently a Professor at the University of Electro-Communications (UEC Tokyo). Since 2008, he has been a Professor Emeritus at UEC Tokyo and is with the Advanced Algorithms Research Laboratory, UEC Tokyo. He was also a Professor at the Research and Development Initiative, Chuo University in 2008–2011. His research interests include combinatorial optimization, algorithmic learning theory, and theory of automata and formal languages. He served as the Director of Information Processing Society of Japan (IPSJ), Chair of Computer Science Domain of IPSJ, PC Chair of ALT 2005, Conference Chair of ICGI 2006, and PC Co-Chair of WALCOM 2015. Dr. Tomita was awarded the Funai Information Technology Prize, Theoretical Computer Science Top Cited Article 2005–2010 for Ref. [30], and others. This article [30] was also elected as the TCS Top Cited Article 2006 at 40th Anniversary of TCS in 2015. His book chapter [33] was downloaded over 10,000 times. He is a Fellow of IPSJ and IEICE.

**Sora Matsuzaki** received his B.Eng. in Informatics and Network Engineering at the University of Electro-Communications in 2017. He is presently a graduate student in the Graduate School of Informatics and Engineering at the same university. He is working on algorithms for the maximum clique problem.

**Atsuki Nagao** received B.Eng. and M.Info. and Dr. of Informatics degrees from Kyoto University in 2010, 2012, and 2015, respectively. From 2015 to 2017, he was a project researcher in the Graduate School of Informatics and Engineering, the University of Electro-Communications. Since 2017, he has been an Assistant Professor in the Department of Computer and Information Science, Seikei University. He is a member of IEICE. He has majored in computational complexity and log-spaced algorithms.

**Hiro Ito** received B.Eng., M.Eng., and Ph.D. degrees in the Department of Applied Mathematics and Physics from the Faculty of Engineering, Kyoto University in 1985, 1987, and 1995, respectively. In 1987–1996, 1996–2001, and 2001–2012, he was a member of NTT Laboratories, Toyohashi University of Technology, and Kyoto University, respectively. Since 2012, he has been a Professor in the Graduate School of Informatics and Engineering at the University of Electro-Communications (UEC Tokyo). He has been engaged in research on discrete algorithms mainly on graphs and networks, discrete mathematics, recreational mathematics, and algorithms for big data. Dr. Ito is a member of IEICE, the Operations Research Society of Japan, the Information Processing Society of Japan, and the European Association for Theoretical Computer Science.

**Mitsuo Wakatsuki** was born in Tokyo, Japan, 1965. He received his Dr.Eng. degree from the University of Electro-Communications in 1993. He is now an Assistant Professor in the Department of Informatics, the Graduate School of Informatics and Engineering at the University of Electro-Communications. His research interests include formal language theory and computational learning theory. Dr. Wakatsuki is a member of IEICE and JSAI.