IEICE TRANS. INF. & SYST., VOL.E96–D, NO.6 JUNE 2013

PAPER

A Simple and Faster Branch-and-Bound Algorithm for Finding a Maximum Clique with Computational Experiments*

Etsuji TOMITA[†,††,†††a)], ***Fellow***, **Yoichi SUTANI**[†,††††], **Takanori HIGASHI**[†,†††††], ***Nonmembers***, ***and*** **Mitsuo WAKATSUKI**[†,††††††], ***Member***

SUMMARY Many problems can be formulated as maximum clique problems. Hence, it is highly important to develop algorithms that can find a maximum clique very fast in practice. We propose new approximate coloring and other related techniques which markedly improve the run time of the branch-and-bound algorithm MCR (J. Global Optim., 37, pp.95–111, 2007), previously shown to be the fastest maximum-clique-finding algorithm for a large number of graphs. The algorithm obtained by introducing these new techniques in MCR is named MCS. It is shown that MCS is successful in reducing the search space quite efficiently with low overhead. Extensive computational experiments confirm the superiority of MCS over MCR and other existing algorithms. It is faster than the other algorithms by orders of magnitude for several graphs. In particular, it is faster than MCR for difficult graphs of very high density and for very large and sparse graphs, even though MCS is not designed for any particular type of graph. MCS can be faster than MCR by a factor of more than 100,000 for some extremely dense random graphs. This paper demonstrates in detail the effectiveness of each new techniques in MCS, as well as the overall contribution.

key words: *maximum clique, branch-and-bound, approximate coloring, computational experiments*

1. Introduction

In an undirected graph G, a *clique* is a subgraph of G, in which all pairs of vertices are adjacent to each other. Finding a maximum clique in a graph is an NP-hard problem, and it is difficult to obtain the exact solution efficiently [6]. It is also difficult to obtain even a satisfactory approximate solution [13]. Nevertheless, we have many practical problems that can be formulated as maximum clique problems. Such examples abound in bioinformatics [1]–[4], [8], [24], [49], drug design [8], pattern recognition and image processing [27], [2], [16], clustering [52], data mining [23], design of quantum circuits [26], design of DNA and RNA sequences for biomolecular computation [19], coding theory [35], design of wireless networks [22], and many others [6].

To date, considerable theoretical progress has been made in the development and analysis of exact algorithms for finding a maximum clique or a maximum independent set (which is a maximum clique of the complementary graph), see e.g., [41], [11], and [31]. However, we should note that an algorithm with theoretically better *order* does not necessarily run faster in practice, see e.g., [34]. Therefore, it is required to develop exact maximum-clique-finding algorithms that run very fast *in practice*. DIMACS workshop on Implementation Challenge for Cliques together with Coloring and Satisfiability was held to address this issue [17].

One standard approach to developing fast algorithms is based on the branch-and-bound method, where the most important point is how to reduce the search space efficiently with *low overhead*. In fact, a significant reduction of the search space is rather easily achieved with high overhead, but it would result in an increase of the overall running time. We developed a branch-and-bound algorithm that is referred to as MCR [47]; in this algorithm, very *simple* approximate coloring and ordering of vertices are employed to reduce the search space. Here, it is very important to choose an appropriate trade-off between the time required for approximate coloring along with ordering of vertices on the one hand and the reduction in the search space thereby achieved on the other. Hence, *simplicity* is very important to *make the overhead as low as possible*. It was shown in computational experiments that MCR clearly outperformed other existing algorithms in finding a maximum clique for a large number of graphs. However, it is not sufficiently fast to solve large practical problems. Hence, much faster algorithms are still in great demand.

In this paper, we propose a new approximate coloring that can play a crucial role in the branch-and-bound algorithm. Subsequently, we introduce a new adjunct ordered

Manuscript received July 26, 2012.
Manuscript revised January 23, 2013.

[†]The authors are with the Advanced Algorithms Research Laboratory, The University of Electro-Communications, Chofu-shi, 182–8585 Japan.
[††]The author is with JST ERATO Minato Discrete Structure Manipulation System Project, Sapporo-shi, 060–0814 Japan.
[†††]The author is also with Tokyo Institute of Technology, Tokyo, 152–8550 Japan.
[††††]The author is with Sony Corporation, Atsugi-shi, 243–0014 Japan.
[†††††]The author is with Japan Systems Co., Ltd., Tokyo, 151–8404 Japan.
[††††††]The author is with the Graduate School of Informatics and Engineering, The University of Electro-Communications, Chofu-shi, 182–8585 Japan.

*A preliminary version of this paper was presented at WALCOM 2010. This work was supported in part by Grants-in-Aid for Scientific Research Nos. 16300001, 19500010, 21300047, 22500009 and 25330009 from the Ministry of Education, Culture, Sports, Science and Technology, Japan, and by a Special Grant for the Strategic Information and Communications R&D Promotion Programme (SCOPE) Project from the Ministry of Internal Affairs and Communications, Japan. It was also supported by the Kayamori Foundation of Informational Science Advancement.

a) E-mail: tomita@ice.uec.ac.jp

DOI: 10.1587/transinf.E96.D.1286

set of vertices for approximate coloring. Next, we present a new technique for reconstructing the adjacency matrix of a graph. This technique is clearly effective when the graph is very large. The algorithm that is obtained by introducing these new techniques into MCR is named MCS. While MCS inherits the *simplicity* of MCR to a large extent, MCS is much more efficient in reducing the search space. The large reduction in size of search spaces for MCS compared to MCR is attributed to the new approximate coloring together with the adjunct ordered set of vertices for approximate coloring introduced in MCS. The resulting overhead in MCS is still low due to the *simplicity* of the newly introduced techniques. Extensive computational experiments have shown that MCS is remarkably faster than MCR and other algorithms. MCS is faster than other algorithms by orders of magnitude for several graphs. In particular, it is faster than MCR for difficult graphs with very high density and for very large and sparse graphs, even though MCS is not designed for any particular type of graph. MCS can be faster than MCR by a factor of more than 100,000 for some extremely dense random graphs.

We describe MCR briefly in Sect. 3, but the reader is advised to refer to [47] for further details. The preliminary versions of this paper appeared in [39] and [48]. This paper demonstrates the details of the effectiveness of the individual and overall contributions of the newly applied techniques in MCS.

2. Definitions and Notation

(1) We consider a simple undirected graph $G = (V, E)$ with a finite set V of vertices and a finite set E of edges that comprises *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 it 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 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 a subgraph *induced* by R. 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. The largest clique in a graph is called a *maximum clique*, and the number of vertices in a maximum clique in an induced subgraph $G(R)$ is denoted by $\omega(R)$.

3. Maximum Clique Algorithm MCR

3.1 Branch-and-Bound Algorithm

The basic branch-and-bound algorithm MCR [47] begins with a small clique and continues finding larger and larger cliques until one is found that can be verified to have the maximum size. To be more precise, 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 may be added to Q. We begin the algorithm by letting $Q := \emptyset$, $Q_{max} := \emptyset$, and $R := V$ (the set of all vertices). We select a certain vertex p from R, 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, Q constitutes a *maximal* clique. If Q is maximal and $|Q| > |Q_{max}|$ holds, we replace Q_{max} 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 $R = \emptyset$.

3.2 Greedy Approximate Coloring

In order to prune unnecessary searching, we used *greedy approximate coloring* [42], [12], [43] of the vertices in MCR. 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 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 the principal *bounding condition*.

After *Numbers* (*Colors*) are assigned to all vertices in R, we sort the vertices in ascending order with respect to their *Numbers*. We refer to the numbering and sorting procedure as NUMBER-SORT [47]. In each step, select a vertex p in R, beginning from the last (rightmost) vertex and ending at the first (leftmost) vertex. As the result, a vertex with the *maximum Number* is selected in a constant time in each step. This is the reason why we sort the vertices in R with respect to their *Numbers*.

Let $maxno := \mathrm{Max}\{\, No[r] \mid r \in R\}$ and $C_i := \{r \in R \mid No[r] = i\}$, where $i = 1, 2, \ldots, maxno$. In other words, C_i is a set of vertices whose *Number* (*Color*) is i. Thus, when the NUMBER-SORT has been applied to R, we have that $R = C_1 \cup C_2 \cup \ldots \cup C_{maxno}$, where the vertices in R are *ordered* in a manner such that first appear the vertices in C_1, and then the vertices in C_2 follow, and so on.

3.3 Initial Sorting and Initial Numbering

In the first stage of algorithm MCQ [45], which is a predecessor of MCR, vertices are sorted in descending order with respect to their degrees and are assigned simple initial *Numbers*. At the beginning of MCR, vertices are sorted and assigned initial *Numbers* in a similar but more sophisticated manner. To be more precise, the steps from {SORT} to just above EXPAND(V, No) in Fig. 4 (Algorithm MCR) in [47] is named *EXTENDED INITIAL SORT-NUMBER* to V.

See Sect. 3.3 (pp.628–630) of [49] for an example run

of MCR.

4. New Algorithm

4.1 New Approximate Coloring

Approximate coloring is generally quite effectively used in branch-and-bound algorithms for finding a maximum clique [12], [43], [45], [47]. Here, we should note that the *minimization* of the number of colors is *not* necessarily most important. It is more important to *reduce the number of vertices from which searching is necessary*. In this paper, we propose a new approximate coloring following greedy approximate coloring in Sect. 3.2 along this line [14]. Because of the *bounding condition* mentioned in Sect. 3.2, if $No[r] \leq |Q_{max}| - |Q|$, then it is not necessary to search from vertex r. The number of vertices to be searched can be reduced if the *Number* $No[p]$ of vertex p for which $No[p] > |Q_{max}| - |Q|$ can be changed to a value that is less than or equal to $|Q_{max}| - |Q|$. When we encounter such vertex p with $No[p] > |Q_{max}| - |Q|$, we attempt to change its *Number* in the following manner. Let No_p denote the original value of $No[p]$.

[Re-NUMBER p]
0) Let $No_{th} := |Q_{max}| - |Q|$. ($No_{th}$ stands for $No_{threshold}$.)
1) Attempt to find a vertex q in $\Gamma(p)$ such that $No[q] = k_1 \leq No_{th}$, with $|\Gamma(p) \cap C_{k_1}| = 1$. (That is, $\Gamma(p) \cap C_{k_1} = \{q\}$.)
2) If such q is found, then attempt to find *Number* k_2 ($> k_1$) such that no vertex in $\Gamma(q)$ has *Number* k_2. (This is possible if $\Gamma(q) \cap C_{k_2} = \emptyset$ but each vertex $r \in C_{k_2}$ is adjacent to some vertex in $C_{k_1} - \{q\}$. Here, it should be that $k_2 > k_1$ because if $k_2 < k_1$ then q should have been Numbered by k_2 ($< k_1$).)
3) If such number k_2 is found, then change the *Number* of q and p so that $No[q] = k_2$ and $No[p] = k_1$.
(If no vertex q with such *Number* k_2 is found, 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 ($< No_{th}$), see Fig. 1; thus, *it is no longer necessary to search from p*.

The exact procedure Re-NUMBER is shown in Fig. 2. To save time, we apply it only when $No[p] = maxno$. The new approximate coloring is described in the first part of Fig. 3 under the heading {NUMBER}; it can be seen that

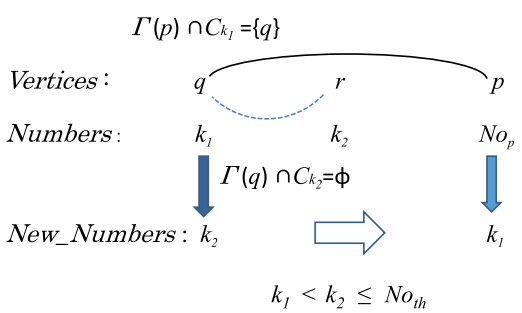

Fig. 1 ReNumbering.

procedure Re-NUMBER($p, No_p, No_{th}, C_1, C_2, ..., C_{maxno}$)
begin
  for $k_1 := 1$ **to** $No_{th} - 1$ **do**
    if $|\Gamma(p) \cap C_{k_1}| = 1$ **then**
      $q :=$ the element in $\left(\Gamma(p) \cap C_{k_1}\right)$;
      for $k_2 := k_1 + 1$ **to** No_{th} **do**
        if $\Gamma(q) \cap C_{k_2} = \emptyset$ **then**
          {Exchange the *Numbers* of p and q.}
          $C_{No_p} := C_{No_p} - \{p\}$;
          $C_{k_1} := \left(C_{k_1} - \{q\}\right) \cup \{p\}$;
          $C_{k_2} := C_{k_2} \cup \{q\}$;
          return
        fi
      od
    fi
  od
end { of Re-NUMBER}

Fig. 2 Procedure Re-NUMBER.

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

      { - Re-NUMBER starts - }
      $No_{th} := |Q_{max}| - |Q|$;
      if $(k > No_{th})$ **and** $(k = maxno)$ **then**
        Re-NUMBER($p, k, No_{th}, C_1, C_2, ..., C_{maxno}$) ;
        if $C_{maxno} = \emptyset$ **then**
          $maxno := maxno - 1$
        fi
      fi
      { - Re-NUMBER ends - }

  od
  {SORT (vertices in R in ascending order w.r.t. their *Numbers*)}
    $i := |V_a|$;
    if $No_{th} < 0$ **then** $No_{th} := 0$ **fi**
    for $k := maxno$ **downto** $No_{th} + 1$ **do**
      for $j := |C_k|$ **downto** 1 **do**
        $R[i] := C_k[j]$;
        $No[R[i]] := k$;
        $i := i - 1$
      od
    od
    if $i \neq 0$ **then**
      $R[i] := C_{k-1}[|C_{k-1}|]$;
      $No[R[i]] := No_{th}$
    fi
end { of Re-NUMBER-SORT }

Fig. 3 Procedure Re-NUMBER-SORT.

Re-NUMBER follows the conventional greedy approximate coloring. The second part of Fig. 3, under the heading {SORT}, describes the sorting of the vertices in R in ascending order with respect to their *Numbers* (Refer to the end of Sect. 3.2). Note that as shown in Fig. 3, vertex r with $No[r] \leq No_{th}$ need not be sorted since the searching operation need not begin from r according to the *bounding condition*.

In Fig. 3, assume that V_a is identical to R for a while (until V_a is introduced in Sect. 4.2).

We employ the new procedure Re-NUMBER-SORT (in Fig. 3) instead of the procedure NUMBER-SORT used in MCR [47] in order to make more effective use of the bounding condition.

The time complexity of Re-NUMBER-SORT is $O(|R|^3)$, while that of NUMBER-SORT [47] is $O(|R|^2)$.

4.2 Adjunct Ordered Set of Vertices for Approximate Coloring

As noted in [12], [43], [9], [45], and [47], the ordering of vertices is crucial in algorithms for finding a maximum clique. The result of approximate coloring greatly depends on the order of vertices because *sequential* coloring is the main component in the procedure. In MCR, the vertices are sorted in descending order mainly with respect to their degrees. When *Numbering* procedures are applied, the vertices are sorted in ascending order with respect to their *Numbers*, and the initial order of the vertices with the same *Number* is *inherited* in the subsequent subproblems [45], [47]. However, the application of *Re-NUMBER*, which is described in Sect. 4.1, changes the *Numbers* of the vertices, thereby making the vertices disordered with respect to their degrees. We can reduce the search space by *sorting* vertices in R *in descending order with respect to their degrees* before every application of approximate coloring. That is, the reduction of the search space is most effective if *the minimum possible Number* is assigned to *a vertex with the maximum degree* in each step of greedy approximate coloring [33]. However, the sorting of vertices is a computational burden and reduces the overall running time only for *dense* graphs [33]. The aim of the present study is to develop a simple and faster algorithm whose use is not confined to any particular type of graph. So, in addition to the ordered set R of vertices, we simply introduce a new particular *adjunct ordered set* V_a of vertices that preserves the order of the vertices *sorted in descending order with respect to their degrees in the first stage*. The adjunct ordered set V_a of vertices was first applied in [37]. We apply the procedure Re-NUMBER-SORT shown in Fig. 3 to the vertices in V_a, beginning from the first (leftmost) vertex and ending at the last (rightmost) vertex. Thus, we can avoid the undesirable effect of Re-NUMBER.

As mentioned in Sect. 3.1, we select a vertex in the ordered set R for *searching*, beginning from the last (rightmost) vertex and continuing up to the first (leftmost) vertex.

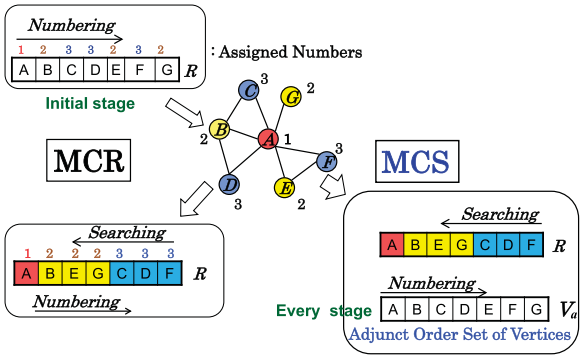

Fig. 4 Adjunct ordered set V_a of vertices for approximate coloring.

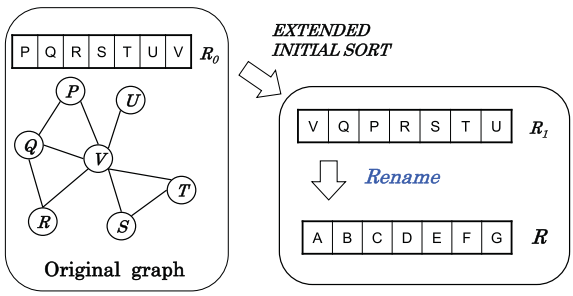

Fig. 5 Reconstruction of the adjacency matrix.

For an example graph consisting of vertices $A, B, \ldots, G$ in the center of Fig. 4, the ordered sets of vertices V_a and R are as shown in MCS part.

Based on the analysis of our previous work [33], we succeeded in improving our earlier maximum-clique-finding algorithms [12], [43] by *adaptively* applying sorting and/or numbering (coloring) of vertices [20], where [12] and [43] are the predecessors of MCQ [45], MCR [47], and MCS. In a similar way in [20], but independently, Konc and Janežič succeeded in improving MCQ [45] to have MaxCliqueDyn (MCDyn for short) [21] that was published after MCR [47].

4.3 Reconstruction of the Adjacency Matrix

Each graph is stored as an adjacency matrix in the computer memory. Sequential numbering in Re-NUMBER-SORT is carried out according to the initial order of vertices in the adjunct ordered set V_a, as described in Sect. 4.2. 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 of MCR. See Fig. 5 for an example. The graph in the center of Fig. 4 is the result of this reconstruction of the adjacency matrix.

The above-mentioned reconstruction of the adjacency matrix results in a more effective use of the cache memory since it facilitates the use of localized memory.

4.4 Algorithm MCS

The new algorithm obtained by introducing the techniques described in Sects. 4.1–4.3 into MCR is named MCS and is shown in Fig. 6.

Note here it could happen, in {SORT} part of Fig. 3, that $maxno < No_{th} + 1$, hence $maxno < |Q_{max}| - |Q| + 1$, *i.e.*, $|Q| + maxno \leq |Q_{max}|$, thus, for-loop (sorting) is not executed. In this case, in EXPAND() in Fig. 6, $|Q| + No[p] > |Q_{max}|$ is false, then 8 lines following **if** $|Q| + No[p] > |Q_{max}|$ **then** is skipped, and the above for-loop (sorting) has no significance.

4.5 Effectiveness of the Reduction of the Search Space and the Running Time

We confirm the effectiveness of the algorithm MCS in reducing the search space. Some characteristic results of computational experiments conducted under the conditions described in Sect. 5 (Computational experiments) for MCR and MCS are listed in Table 1.

Table 1 lists the number of branches, that is, the total number of EXPAND() calls excluding the first call, of MCR and MCS for random graphs r200.9 – r500.994 and several DIMACS benchmark graphs in the leftmost column. The random graphs r200.9, r200.95, and r200.98 are graphs

```
procedure MCS (G = (V, E))
begin
  global Q := ∅; global Q_max := ∅;
  {EXTENDED INITIAL SORT-NUMBER}
  Apply EXTENDED INITIAL SORT-NUMBER to V (see Sect. 3.3);
  Reconstruct the adjacency matrix as described in Sect. 4.3;
  EXPAND (V, V, No);
  output Q_max {Maximum clique}
end { of MCS }

procedure EXPAND(V_a, R, No)
begin
  while R ≠ ∅ do
    p := the last vertex in R
         (i.e., a vertex with the maximum Number in R);
    if |Q| + No[p] > |Q_max| then
      Q := Q ∪ {p};
      V_p := V_a ∩ Γ(p); {preserving the order}
      if V_p ≠ ∅ then
        Re-NUMBER-SORT(V_p, newR, newNo);
        {The initial values of newR and newNo have no significance}
        EXPAND(V_p, newR, newNo)
      else if |Q| > |Q_max| then Q_max := Q fi
      fi
    else return
    fi
    Q := Q − {p};
    R := R − {p};
    V_a := V_a − {p} {preserving the order}
  od
end { of EXPAND }
```

Fig. 6 Algorithm MCS.

Table 1 Comparison of branches.

Graph		$Branches \times 10^{-3}$			CPU time
Name	ω	MCR	MCS	$(MCR/MCS)_b$	$(MCR/MCS)_t$
r200.9	40-44	97,627	6,310	15	9
r200.95	58-66	104,801	2,735	38	22
r200.98	90-103	2,357	4	589	155
r300.8	28-29	253,185	48,685	5.2	3.2
r300.98	120	4.03×10^6	31,619	127	108
r500.7	22-23	675,344	227,922	3.0	2.1
r500.994	263	$> 4.29 \times 10^6$	70	$> 61,286$	$> 256,410$
brock200_1	21	482	144	3.3	2.0
brock400_1	27	329,599	89,389	3.7	2.6
brock400_4	33	114,925	28,644	4.0	2.6
brock800_1	23	2,715,369	1,091,680	2.5	1.9
MANN_a27	126	38	9	13	4.2
MANN_a45	345	2,952	225	13	11
p_hat300-3	36	1,546	235	6.6	4.3
p_hat500-2	36	408	64	6.4	4.4
p_hat500-3	50	138,300	7,923	18	12
p_hat700-2	44	4,115	324	12.7	7.9
p_hat700-3	62	3,733,665	88,168	42	29
p_hat1000-2	46	197,147	12,618	16	11
san200_0.7_1	30	2.94	0.70	4.2	2.6
san200_0.9_1	70	195	17	11.5	5.5
san200_0.9_2	60	595	37	16	11
san400_0.7_3	22	410	124	3.3	2.8
san400_0.9_1	100	74	2	37	28
san1000	15	230	82	2.8	2.3
sanr200_0.9	42	40,470	3,471	11.7	7.0
sanr400_0.7	21	89,124	28,513	3.1	2.1
gen200_p0.9_44	44	583	35	17	12
gen200_p0.9_55	55	2,335	112	21	13
gen400_p0.9_55	55	$> 4.29 \times 10^6$	2,894,935	> 1.5	100
gen400_p0.9_65	55	$> 4.29 \times 10^6$	3,332,982	> 1.3	> 131

Table 2 Comparison of algorithms (Branches).

Graph		*Branches* $\times 10^{-3}$				
Name	ω	MCR	MCR-R	MCR*	MCR*-R	MCS
r200.9	40-44	97,627	18,401	20,660	6,310	6,310
r300.8	28-29	253,185	89,147	116,636	48,685	48,685
r500.7	22-23	675,344	346,365	422,930	227,922	227,922
brock200_1	21	482	218	296	144	144
brock400_1	27	329,599	139,299	182,920	89,389	89,389
brock400_4	33	114,925	47,783	62,859	28,644	28,644
brock800_1	23	2,715,369	1,471,741	1,905,718	1,091,680	1,091,680
MANN_a27	126	38	$> 4.29 \times 10^6$	38	9	9
MANN_a45	345	2,952	$> 4.29 \times 10^6$	2,952	225	225
p_hat300-3	36	1,546	559	497	235	235
p_hat500-3	50	138,300	34,098	18,185	7,923	7,923
p_hat700-2	44	4,115	1,130	701	324	324
p_hat700-3	62	3,733,665	692,368	201,949	88,168	88,168
p_hat1000-2	46	197,147	55,848	25,648	12,618	12,618
san200_0.9_1	70	195	48	95	17	17
san200_0.9_2	60	595	76	200	37	37
san400_0.7_3	22	410	211	244	124	124
san400_0.9_1	100	74.0	2.4	20.0	2.1	2.1
san1000	15	230	162	193	82	82
sanr200_0.9	42	40,470	7,496	11,775	3,471	3,471
sanr400_0.7	21	89,124	43,206	54,622	28,513	28,513

Table 3 Comparison of algorithms (CPU time).

Graph		CPU time [sec]				
Name	ω	MCR	MCR-R	MCR*	MCR*-R	MCS
r200.9	40-44	647	197	158	82	74
r300.8	28-29	1,264	712	637	422	394
r500.7	22-23	3,268	2,552	2,173	1,741	1,539
brock200_1	21	1.72	1.26	1.16	0.90	0.86
brock400_1	27	1,771	1,157	1,057	748	693
brock400_4	33	639	410	378	262	248
brock800_1	23	17,789	13,991	12,900	10,204	9,347
MANN_a27	126	2.54	$> 10^5$	2.56	0.82	0.78
MANN_a45	345	3,090	$> 10^5$	3,089	314	281
p_hat300-3	36	10.82	6.08	4.07	2.65	2.54
p_hat500-3	50	1,788	649	270	165	150
p_hat700-2	44	44.42	18.27	9.28	6.34	5.60
p_hat700-3	62	68,187	18,401	4,374	2,504	2,392
p_hat1000-2	46	2,434	1,002	376	239	221
san200_0.9_1	70	1.20	0.44	0.70	0.23	0.22
san200_0.9_2	60	4.17	0.70	1.35	0.41	0.41
san400_0.7_3	22	3.60	2.17	2.55	1.46	1.44
san400_0.9_1	100	3.43	0.14	1.00	0.12	0.12
san1000	15	4.82	3.45	4.75	2.20	2.14
sanr200_0.9	42	289	86	94	45	41
sanr400_0.7	21	379	284	248	197	181

Table 4 Effect of reconstruction of the adjacency matrix.

Graph			CPU time [sec]				
n	p	ω	dfmax	MCR	MCR*-R	MCS	MCR*-R/MCS
3,000	0.1	6-7	0.80	0.73	0.73	0.60	1.22
3,000	0.2	9	16.9	13.0	12.3	9.5	1.30
3,000	0.3	11-12	631	360	324	288	1.13
3,000	0.4	14	44,592	18,894	15,902	14,442	1.10
5,000	0.1	7	6.3	5.3	5.4	3.3	1.64
5,000	0.2	9	259	197	193	138	1.40
5,000	0.3	12	14,008	8,668	7,921	5,818	1.36
10,000	0.1	7-8	137	100	100	60	1.67
10,000	0.2	10	9,417	8,055	7,876	4,389	1.79
15,000	0.1	8	793	511	496	327	1.52
20,000	0.1	8	2,665	1,737	1,705	1,179	1.45

Table 5 CPU time [sec] for random graphs. (To be continued.)

Graph			dfmax	MCR		MCS	New	MCDyn
n	p	ω	[17]	[47]			[28]	[21]
	0.5	9-10	0.00140	0.00078		**0.00077**		
	0.6	11-13	0.0041	0.0017		**0.0016**	0.0022	0.0025
	0.7	14-16	0.0180	0.0047		**0.0036**	0.0067	0.0066
100	0.8	19-21	0.140	0.014	·	**0.008**	0.065	0.018
	0.9	29-32	3.67	0.038	○	**0.013**	0.663	0.034
	0.95	39-48	23.736	0.011	·	**0.003**	0.196	0.005
	0.98	56-68	26.5401	0.0012		**0.0009**		
	0.7	16-18	0.359	0.074	·	**0.048**		0.084
	0.8	23	6.88	0.55	○	**0.23**		0.54
150	0.9	36-39	1058.96	5.26	○	**1.00**		3.09
	0.95	50-59	37,436.79	3.94		**0.35**		0.36
	0.98	73-85	$> 10^5$	0.243	★○	**0.006**		
	0.4	9-10	0.0082	0.0049		**0.0045**	0.0067	0.0070
	0.5	11-12	0.038	0.017		**0.015**	0.020	0.028
	0.6	14	0.29	0.09		**0.07**	0.17	0.12
200	0.7	18-19	3.85	0.68	·	**0.41**	3.02	0.77
	0.8	24-27	192.7	12.3	○	**4.5**	147.3	9.9
	0.9	40-44	$> 10^5$	647	○	**74**		250
	0.95	58-66	$> 10^5$	1,272		59		**54**
	0.98	90-103	$> 10^5$	30.9	★★	**0.2**		
	0.4	9-10	0.360	0.028		**0.026**	0.047	0.048
	0.5	12-13	4.88	0.15		**0.13**	0.20	0.24
	0.6	15-16	144.1	1.4		**1.0**	3.5	1.9
300	0.7	19-21	26,236	23	·	**12**	121	26
	0.8	28-29	$> 10^5$	1,264	○	**394**		1,288
	0.9	49		1,475,387	★○	**62,607**		
	0.98	120	$> 10^5$	282,917	★★	**2,623**		
	0.99	154	$> 10^5$	732.49	★★★	**0.23**		
400	0.99	188		$> 9.5 \times 10^6$	★★★	**1,030**		
	0.3	8-9	0.079	0.052		**0.050**	0.083	0.097
	0.4	11	0.65	0.35		**0.31**	0.60	0.52
500	0.5	13-14	9.0	3.6		**2.8**	7.3	5.2
	0.6	17	242	63	·	**40**	183	82
	0.7	22-23	24,998	3,268	○	**1,539**		
	0.994	263	$> 1.5 \times 10^7$	$> 10^7$	★★★★★	**39**		

Entries marked ★★★★★, ★★★, ★★,★○, ★, ○, and · are respectively
at least 100,000, 1,000, 100, 20, 10, 2, and 1.5 times faster than any of the others in the same row.

with 200 vertices and with edge probabilities 0.9, 0.95, and 0.98, respectively. The number of branches specified for r200.9 is the average over 10 graphs, and the number of branches given for r200.95 and r200.98 is the average over 100 graphs. The second column (ω) lists the ranges of the maximum clique sizes obtained. In Table 1, the values for graphs with names of the form rn.p ($n = 300, 500$ and $p = 0.98, 0.994$) are obtained from one random graph with n vertices and with edge probability p. The number of branches given for r300.8 and r500.7 is the average over 10 graphs. The number of branches is related to the size of the search space. (Note that $4.29 \times 10^9 = 2^{32}$.) The fifth column $(MCR/MCS)_b$ lists the ratio of the number of branches of MCR to that of MCS. The ratio of the CPU time required by MCR to that of MCS for each graph is given in the last column $(MCR/MCS)_t$ for reference and has been obtained from Tables 5 and 6 in Sect. 5. Table 1 confirms that MCS is quite successful in reducing the search space. In addition, we can see that the reduction of the search space by MCS effectively contributes to the reduction of the running time. We have confirmed that the search space of MCS is considerably smaller than that of MCR for all graphs in Sect. 5.

4.6 Details of the Effectiveness of MCS

We examine the individual contributions given in Sects. 4.1–4.3 for the effectiveness of MCS.

We give names for intermediate algorithms between MCR and MCS as follows.

MCR-R: improved MCR obtained by introducing only the technique in Sect. 4.1 [14].

MCR*: improved MCR obtained by introducing only the technique in Sect. 4.2 [37].

MCR*-R: improved MCR obtained by introducing the combination of the techniques in Sects. 4.1 and 4.2 [38].

Table 5 (Continued.)

Graph			dfmax	MCR		MCS	New	MCDyn
n	p	ω	[17]	[47]			[28]	[21]
	0.2	7-8	0.170	0.134		**0.129**	0.208	0.344
	0.3	9-10	1.98	1.28		**1.15**	1.64	2.08
1,000	0.4	12	33.3	16.1		**13.2**	23.2	30.1
	0.5	15	1,107	395		**290**		585
	0.6	19-20	106,776	24,986	·	**15,317**		
	0.66	23		555,089	○	**275,964**		
	0.998	618		$> 10^7$	★★★★★	**46**		
1,500	0.998	997		$> 10^7$	★★★★★	**13**		
2,000	0.9995	1,453		$> 10^7$	★★★★★	**61**		
	0.1	6-7	0.8	0.7		**0.6**		
3,000	0.2	9	16.9	13.0		**9.5**		
	0.3	11-12	631	360		**288**		
	0.4	14	44,592	18,894		**14,442**		
	0.1	7	6.3	5.3	·	**3.3**		
5,000	0.2	9	259	197		**138**		
	0.3	12	14,008	8,668		**5,818**		
10,000	0.1	7-8	137	100	·	**60**		
	0.2	10	9,417	8,055	·	**4,389**		
15,000	0.1	8	793	511	·	**327**		
20,000	0.1	8	2,665	1,737		**1,179**		

Entries marked ★★★★★, ○, and · are respectively
at least 100,000, 2, and 1.5 times faster than any of the others in the same row.

The results of the comparison of these algorithms for some of the graphs in Table 1 are shown in Tables 2 and 3. (The numbers of branches of MCR*-R and MCS are exactly the same.)

Table 2 shows that MCR-R and MCR* are successful in reducing the search space of MCR for most of the graphs, but with some exception (MANN_a27, MANN_a45). Note that, for some graphs (r200.9, ..., brock800_1, sanr400_0.7)), MCR* is faster than MCR-R while the number of branches of MCR* is larger than that of MCR-R. This is because of the overhead of time-consuming Re-NUMBER procedure. As a result of combination of techniques in Sects. 4.1–4.2, MCR*-R is successful to reduce the search space of MCR efficiently for all the graphs in Table 2, and hence MCR*-R is faster than MCR for all of these graphs.

The effect of the reconstruction of the adjacency matrix in Sect. 4.3 (MCR*-R versus MCS) is not significant in these Tables since these graphs are not large in the number of vertices, so we compared MCR*-R and MCS for *large* graphs. The result is shown in Table 4, where n stands for the number of vertices and p stands for the edge probability. They are the same as the corresponding ones in Table 5 (Sect. 5.1). The column MCR*-R/MCS shows the ratio of each CPU time required by MCR*-R to that by MCS for each graph.

The table demonstrates the clear effectiveness of this technique for large graphs that have more than or equal to 5,000 vertices. Such effect is larger than that obtained by MCR*-R over MCR for these graphs. Note that the effectiveness of the reconstruction of the adjacency matrix depends not only on the tested graphs but also on the computers used.

In summary, the effectiveness of MCS over MCR is obtained as the *combination* of all the techniques in Sects. 4.1–4.3.

5. Computational Experiments

We carried out computational experiments in order to demonstrate the overall superiority of MCS over MCR. Both MCR and MCS were implemented in exactly the same manner in the programming language C. The computer used, which had a Linux operating system, is described in Appendix. We also executed the DIMACS benchmark program dfmax [17], [18] as a standard. The computation times for other algorithms are calibrated using the ratios shown in Appendix.

5.1 Results for Random Graphs

Random graphs are generated for each pair of n (number of vertices) and p (edge probability) listed in Table 5. These graphs are generated such that there exists an edge with probability p for each pair of vertices. The average CPU times [sec] required to solve these graphs when using dfmax, MCR, and MCS are listed in Table 5. The CPU times

Table 6 CPU time [sec] for DIMACS benchmark graphs. (To be continued.)

Graph				dfmax	MCR		MCS	New	ILOG	MCDyn
Name	n	*density*	ω	[17]	[47]			[28]	[30]	[21]
brock200_1	200	0.754	21	14.53	1.72	·	**0.86**	12.12	7.94	1.39
brock200_2	200	0.496	12	0.028	0.011		**0.010**	0.011	0.215	0.016
brock200_3	200	0.605	15	0.210	0.055		**0.042**	0.098	0.637	0.066
brock200_4	400	0.658	17	0.90	0.22	·	**0.14**	0.22	1.58	0.22
brock400_1	400	0.75	27	22,051	1,771	·	**693**		8,401	1,050
brock400_2	400	0.75	29	13,519	726	·	**297**		5,860	465
brock400_3	400	0.75	31	14,795	1,200	·	**468**		3,316	851
brock400_4	400	0.75	33	10,633	639	·	**248**		4,483	476
brock800_1	800	0.65	23	$> 10^5$	17,789		**9,347**		$> 10,667$	12,807
brock800_2	800	0.65	24	$> 10^5$	16,048		**8,368**		$> 10,668$	12,060
brock800_3	800	0.65	25	91,031	10,853		**5,755**		$> 10,669$	8,271
brock800_4	800	0.65	26	78,737	7,539	·	**3,997**		$> 10,670$	6,411
c-fat200-1	200	0.077	12	0.0005	0.0006		0.0007	0.0022		**0.0004**
c-fat200-2	200	0.163	24	0.0006	0.0011		0.0012	0.0022		**0.0004**
c-fat200-5	200	0.426	58	268	**0.0017**		0.0020	1.7433		0.0027
c-fat500-1	500	0.036	14	0.0031	0.0028		0.0033	0.0156		· **0.0018**
c-fat500-2	500	0.073	26	0.0032	0.0046		0.0054	0.0179		**0.0027**
c-fat500-5	500	0.186	64	1.713	0.013		0.013	2,331		· **0.008**
c-fat500-10	500	0.374	126	$> 10^5$	0.024		0.026	**0.016**	0.023	0.031
hamming6-2	64	0.905	32	0.01064	**0.000074**		0.000088	0.002232		0.000446
hamming6-4	64	0.349	4	0.00015	0.000076		0.000097	0.002232		**0**
hamming8-2	256	0.969	128	$> 10^5$	**0.0022**		0.0025	0.0089		0.0161
hamming8-4	256	0.639	16	1.85	0.22		0.20	0.19	3.10	**0.13**
hamming10-2	1,024	0.990	512	$> 10^5$	**0.17**		0.19	0.56	0.77	5.92
johnson8-2-4	28	0.556	4	0.00004	0.00002		0.00002	0.00223		**0**
johnson8-4-4	70	0.768	14	0.0045	○ **0.0004**	○	**0.0004**	0.0022		0.0009
johnson16-2-4	120	0.765	8	0.75	0.14		0.13	○ **0.062**	2.81	0.608
keller4	171	0.649	11	0.374	0.028		**0.025**	0.112	0.370	0.036
MANN_a9	45	0.927	16	0.0413	○ **0.0001**	○	**0.0001**	0.0022		0.0004
MANN_a27	378	0.990	126	$> 10^5$	2.5	○	**0.8**	$> 2,232$	13.7	6.8
MANN_a45	1,035	0.996	345	$> 10^5$	3,090	★	**281**		$> 10,670$	8,069
p_hat300-1	300	0.244	8	0.0067	**0.0048**		0.0050	0.0089	0.0814	0.0049
p_hat300-2	300	0.489	25	0.63	0.03		**0.02**	0.22	0.44	0.06
p_hat300-3	300	0.744	36	779.7	10.8	○	**2.5**		30.2	8.1
p_hat500-1	500	0.253	9	0.051	0.031		**0.030**	0.065	1.704	0.037
p_hat500-2	500	0.505	36	132.9	3.1	○	**0.7**	95.7	24.2	2.5
p_hat500-3	500	0.752	50	$> 10^5$	1,788	○	**150**		9,441	660
p_hat700-1	700	0.249	11	0.20	0.11		**0.10**	0.15	4.45	0.17
p_hat700-2	700	0.498	44	5,299.9	44.4	○	**5.6**		189.5	22.7
p_hat700-3	700	0.748	62	$> 10^5$	68,187	○	**2,392**		$> 10,670$	12,128
p_hat1000-1	1,000	0.245	10	1.05	0.60		**0.49**	1.30	20.59	0.88
p_hat1000-2	1,000	0.489	46	$> 10^5$	2,434	○	**221**		12,478	767
p_hat1500-1	1,500	0.253	12	10.1	5.1		**3.9**		356.2	6.9
p_hat1500-2	1,500	0.506	65	$> 10^5$	722,733	★○	**16,512**		$> 10,670$	$> 19,286$

Entries marked ★○, ★, ○, and · are respectively
at least 20, 10, 2, and 1.5 times faster than any of the others in the same row.

are averaged over 10 random graphs for each pair of n and p. However, when the CPU time [sec] is greater than 10^5, the individual value of the graph, instead of the average, is listed. The CPU times required to solve the graphs with $n \leq 200$ and $p \geq 0.95$ are averaged over 100 graphs because of the large variations in these graphs and the short running time of MCR and MCS. For graphs with $n \geq 300$ and $p \geq 0.9$, the CPU time for only one graph is considered for each pair of n and p (10^5 seconds $\simeq$ 1.16 days, and 10^7 seconds $\simeq$ 116 days). The third column (ω) lists the ranges of the sizes of the maximum cliques obtained.

The calibrated CPU times for New [28] and MCDyn [21] are also listed for reference. The boldface entries indicate the fastest time in the row. In Table 5, it is observed that MCS is faster than MCR for all graphs. MCS is particularly faster than MCR for dense graphs. MCS is the fastest for all the random graphs listed in Table 5, except for that with $[n = 200, p = 0.9]$. For this graph, MCDyn is slightly faster than MCS. The calibrated CPU time by COCR [32] for $[n = 200, p = 0.9]$ is 37 seconds that is the

Table 6 (Continued.)

Graph Name	n	*density*	ω	dfmax [17]	MCR [47]		MCS		New [28]	ILOG [30]	MCDyn [21]
san200_07_1	200	0.700	30	2,578	0.021		**0.008**		0.125	0.267	0.011
san200_07_2	200	0.700	18	16,110	0.007		**0.005**		0.009	0.274	0.013
san200_0.9_1	200	0.900	70	$> 10^5$	1.20		0.22	○	**0.06**	0.77	0.43
san200_0.9_2	200	0.900	60	$> 10^5$	4.2	○	**0.4**		1.0	1.9	1.3
san200_0.9_3	200	0.900	44	42,643	0.16	○	**0.06**			135.3	5.95
san400_0.5_1	400	0.500	13	433	0.022		0.020	○	**0.007**	0.881	0.023
san400_0.7_1	400	0.700	40	$> 10^5$	1.76		**0.54**		$> 2,232$	17.24	0.73
san400_0.7_2	400	0.700	30	$> 10^5$	0.33	·	**0.13**		112.97	50.02	0.26
san400_0.7_3	400	0.700	22	$> 10^5$	3.6	○	**1.4**			202.4	2.9
san400_0.9_1	400	0.900	100	$> 10^5$	3.4	★○	**0.1**			1,259.3	59.3
san1000	1,000	0.502	15	$> 10^5$	4.8		2.1	★	**0.1**	76.1	1.4
sanr200_0.7	200	0.702	18	3.06	0.569	·	**0.338**		3.150	3.185	0.523
sanr200_0.9	200	0.898	42	86,954	289	○	**41**			111	92
sanr400_0.5	400	0.501	13	2.12	0.89		**0.72**		1.48	12.68	1.02
sanr400_0.7	400	0.700	21	2,426	379		**181**			2,325	257
DSJC500.5	500	0.502	13	9.9	4.2		**3.1**				
DSJC1000.5	1,000	0.500	15	1,132	405		**293**				
gen200_p0.9_44	200	0.900	44	48,262	5.39	★	**0.47**				
gen200_p0.9_55	200	0.900	55	9,281.0	15.0	★	**1.2**				
gen400_p0.9_55	400	0.900	55		5,846,951	★★	**58,431**				
gen400_p0.9_65	400	0.900	65		$> 2 \times 10^7$	★★	**151,597**				
gen400_p0.9_75	400	0.900	75		$> 10^7$	★○	**294,175**				
C125.9	125	0.898	34	50.05	0.24	○	**0.06**				
C250.9	250	0.899	44	$> 10^6$	44,214	★	**3,257**				
C2000.5	2,000	0.500	16	292,291	102,571		**71,862**				

Entries marked ★★, ★●, ★○, ★, ○, and · are respectively
at least 100, 50, 20, 10, 2, and 1.5 times faster than any of the others in the same row.

half of that of MCS. The calibrated CPU times by COCR for $[n = 150, p = 0.9]$ and $[n = 200, p = 0.8]$ are 1.16 and 8.7 seconds, respectively, which are larger than those of MCS and smaller than those of MCR [47]. COCR is specially designed for solving the maximum clique problem for *dense* graphs.

For the graphs with $p \geq 0.99$ in Table 5, MCS is faster than MCR by a factor of greater than 100,000. Note here that tested random graphs for New and MCDyn are not exactly the same ones for MCS. Therefore, the difference of the calibrated CPU times could be meaningful if the difference is more than 2 times or so.

Regarding dfmax, it was stated in [18] that "It ... may be hard to beat on sparser graphs, especially random ones." Prior to the development of MCQ [45], dfmax was widely recognized as the fastest maximum clique algorithm for *sparse* graphs, as stated in [10] and [28]. MCQ and its successors are faster than dfmax, even for sparse graphs. MCS is the *only* algorithm that is *more than twice as fast as dfmax* for sparse graphs with 10,000 or more vertices (Table 5).

5.2 Results for DIMACS Benchmark Graphs

Table 6 lists the CPU times required by MCS and other algorithms to solve the DIMACS benchmark graphs [17], where the calibrated CPU times for New [28], ILOG [30], and MCDyn [21] are included for reference. In this table, *density* represents the edge density of the graph. The boldface entries indicate the fastest time among the times obtained within the time limits in the row. From this table, it is confirmed that MCS is faster than MCR and the other algorithms in Table 6, excluding easy graphs which can be solved by MCS in less than 0.3 seconds. The only one exceptional graph is san1000 for which New is the fastest and MCDyn is the second fastest.

MCS is almost always considerably faster than χ+DF [10], COCR [32], MIPO [5], SQUEEZE [7], and Target [36] (see Table 4 in [47]). Although COCR is specially designed to efficiently find a maximum clique in *dense* graphs, the calibrated CPU time of COCR to solve MANN_a27 with high density (*density* = 0.990) is 2.8 seconds, that is 3.5 times larger than that of MCS. Further, MCS is confirmed to be much faster than MC of Wood [51] and CP+SDP of Hoeve [15], as is evident in [30].

In addition, as one of the practical applications in coding theory [35], MCS solved 2dc.1024 (in Graphs From Two-Deletion-Correcting Codes) in 104.7 seconds ($n = 1,024$, *density* $= 0.68, \omega = 16$). MCS is also successfully applied to solve a problem in bioinformatics [24] where preliminary experiments confirmed the superiority of MCS over MCR and MCQ for this problem.

6. Concluding Remarks

Our new algorithm, MCS, retains the *simplicity* of our earlier algorithms while *further reducing the search space* quite efficiently with *low overhead*. This paper has made the effectiveness of the individual and overall contributions of the newly applied techniques in MCS clearer. Note that the effectiveness of MCS depends not only on the tested graphs but also on the computers used, as mentioned in Sect. 4.6.

Our proposed techniques will be useful for generating large maximal cliques [25], [46]. Combining some near-maximum-clique-finding algorithm as a preprocessor of MCS can make MCS more efficient [38]. In [38], an improved very simple near-maximum-clique-finding algorithm in [44] was employed. Improving MCS by adaptively applying sorting and/or numbering (coloring) of vertices along the lines of [20] is our next work. Parallel processing of MCS is also promising [40], [50].

Acknowledgements

We would like to express our sincere gratitude to S. Takahashi for his contribution in an early stage of this work. We thank E. Harley, T. Akutsu, M. Haraguchi, S. Minato, T. Nishino, the reviewers and the editor of this Transactions, and many others for useful comments, kind help, and encouragement. We wish to thank P.M. Pardalos and his colleagues for reviewing our earlier works including [43] and [44] in their surveys [6], [29]. Our earlier works received considerable attention by their reviews. Thanks are also to D.S. Johnson and M.A. Trick for their efforts in organizing the Second DIMACS Implementation Challenge for Cliques, Coloring, and Satisfiability [17]. They made it easier for us to compare the results of different algorithms carried out on various computers.

References

[1] T. Akutsu, M. Hayashida, D.K.C. Bahadur, E. Tomita, J. Suzuki, and K. Horimoto, "Dynamic programming and clique based approaches for protein threading with profiles and constraints," IEICE Trans. Fundamentals, vol.E89-A, no.5, pp.1215–1222, May 2006.

[2] D.K.C. Bahadur, T. Akutsu, E. Tomita, T. Seki, and A. Fujiyama, "Point matching under non-uniform distortions and protein side chain packing based on efficient maximum clique algorithms," Genome Inform., vol.13, pp.143–152, 2002.

[3] D.K.C. Bahadur, E. Tomita, J. Suzuki, K. Horimoto, and T. Akutsu, "Protein side-chain packing problem: A maximum edge-weight clique algorithmic approach." J. Bioinform. and Comput. Biol., vol.3, pp.103–126, 2005.

[4] D.K.C. Bahadur, E. Tomita., J. Suzuki, K. Horimoto, and T. Akutsu, "Protein threading with profiles and distance constraints using clique based algorithms," J. Bioinform. and Comput. Biol., vol.4, pp.19–42, 2006.

[5] E. Balas, S. Ceria, G. Cornuéjols, and G. Pataki, "Polyhedral methods for the maximum clique problem," in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.26, ed. D.S. Johnson and M.A. Trick, pp.11–28, 1996.

[6] I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo, "The maximum clique problem," in Handbook of Combinatorial Optimization, Supplement vol.A, ed. D.-Z. Du and P.M. Pardalos, pp.1–74, Kluwer Academic Publishers, 1999.

[7] J.-M. Bourjolly, P. Gill, G. Laporte, and H. Mercure, "An exact quadratic 0-1 algorithm for the stable set problem," in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.26, ed. D.S. Johnson and M.A. Trick, pp.53–73, 1996.

[8] S. Butenko and W.E. Wilhelm, "Clique-detection models in computational biochemistry and genomics — Invited Review," Eur. J. Oper. Res., vol.173, pp.1–17, 2006.

[9] R. Carraghan and P.M. Pardalos, "An exact algorithm for the maximum clique problem," Operations Research Letters, vol.9, pp.375–382, 1990.

[10] T. Fahle, "Simple and fast: Improving a branch-and-bound algorithm for maximum clique," European Symp. on Algorithms 2002, LNCS 2461, pp.485–498, 2002.

[11] F. Fomin, F.V. Grandoni, and D. Kratsch, "A measure & conquer for the analysis of exact algorithms," J. ACM, pp.25(1)–25(32), 2009.

[12] T. Fujii and E. Tomita, "On efficient algorithms for finding a maximum clique," Tech. Rep. IECE, AL81-113, pp.25–34, 1982.

[13] J. Håstad, "Clique is hard to approximate within $n^{1-\varepsilon}$," Acta Mathematica, vol.182, pp.105–142, 1999.

[14] T. Higashi and E. Tomita, "A more efficient algorithm for finding a maximum clique based on an improved approximate coloring," Tech. Rep. Univ. Electro-Commun., UEC-TR-CAS5, 2006.

[15] W.J. von Hoeve, "Exploiting semidefinite relaxations in constraint programming," Computers & Operations Research, vol.33, pp.2787–2804, 2006.

[16] K. Hotta, E. Tomita, and H. Takahashi, "A view-invariant human face detection method based on maximum cliques," Trans. IPSJ, vol.44, no.SIG14(TOM9), pp.57–70, 2003.

[17] D.S. Johnson and M.A. Trick (eds.), "Cliques, coloring, and satisfiability," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol.26, American Math. Soc., 1996.

[18] http://www.cs.sunysb.edu/~algorith/implement/dimacs/distrib/color/graph/form

[19] S. Kobayashi, T. Kondo, K. Okuda, and E. Tomita, "Extracting globally structure free sequences by local structure freeness," in Proc. Ninth International Meeting on DNA Based Computers, ed. J. Chen and J. Reif, p.206, 2003.

[20] Y. Kohata, T. Nishijima, E. Tomita, C. Fujihashi, and H. Takahashi, "Efficient algorithms for finding a maximum clique," Tech. Rep. IEICE, COMP89-113, pp.1–8, 1990.

[21] J. Konc and D. Janežič, "An improved branch and bound algorithm for the maximum clique problem," MATCH Commun. in Mathematical and in Computer Chemistry, vol.58, pp.569–590, 2007.

[22] E. Liu, Q. Zhang, and K.K. Leung, "Clique-based utility maximization in wireless mesh networks," IEEE Trans. Wireless Commun., vol.10, no.3, pp.948–957, 2011.

[23] T. Matsunaga, C. Yonemori, E. Tomita, and M. Muramatsu, "Clique-based data mining for related genes in a biomedical database," BMC Bioinformatics, vol.10, pp.1–9, 2009.

[24] T. Mori, T. Tamura, D. Fukagawa, A. Takasu, E. Tomita, and T. Akutsu, "A clique-based method using dynamic programming for computing edit distance between unordered trees," J. Computational Biology, vol.19, pp.1089–1104, 2012.

[25] T. Nakagawa and E. Tomita, "An efficient algorithm for generating large maximal cliques," Tech. Rep. IPSJ, 2005-MPS-57, pp.49–52, 2005.

[26] Y. Nakui, T. Nishino, E. Tomita, and T. Nakamura, "On the minimization of the quantum circuit depth based on a maximum clique with maximum vertex weight," Tech. Rep. RIMS, 1325, Kyoto Univ., pp.45–50, 2003.

[27] H. Ogawa, "Labeled point pattern matching by Delaunay triangulation and maximal cliques," Pattern Recognit., vol.19, pp.35–40, 1986.

[28] P.R.J. Östergård, "A fast algorithm for the maximum clique problem," Discrete Appl. Math., vol.120, pp.197–207, 2002.

[29] P.M. Pardalos and J. Xue, "The maximum clique problem," J. Global Optim., vol.4, pp.301–328, 1994.
[30] J.-C. Régin, "Using constraint programming to solve the maximum clique problem," Principles and Practice of Constraint Programming, LNCS 2833, pp.634–648, 2003.
[31] J.M. Robson, "Finding a maximum independent set in time $O(2^{n/4})$," Tech. Rep. 1251-01, LaBRI, Universite Bordeaux, 2001.
[32] E.C. Sewell, "A branch and bound algorithm for the stability number of a sparse graph," INFORMS J. Computing, vol.10, pp.438–447, 1998.
[33] M. Shindo, E. Tomita, and Y. Maruyama, "An efficient algorithm for finding a maximum clique," Tech. Rep. IECE, CAS86-5, pp.33–40, 1986.
[34] M. Shindo and E. Tomita, "A simple algorithm for finding a maximum clique and its worst-case time complexity," Systems and Computers in Japan, vol.21, pp.1–13, Wiley, 1990.
[35] N.J.A. Sloane, "Challenge problems: Independent sets in graphs," http://www.research.att.com/~njas/doc/graphs.html
[36] V. Stix, "Target-oriented branch and bound method for global optimization," J. Global Optim., vol.26, pp.261–277, 2003.
[37] Y. Sutani and E. Tomita, "Computational experiments and analyses of a more efficient algorithm for finding a maximum clique," Tech. Rep. IPSJ, 2005-MPS-57, pp.45–48, 2005.
[38] Y. Sutani, T. Higashi, and E. Tomita, "A more efficient algorithm for finding a maximum clique with an improved approximate coloring," Tech. Rep. Summer LA Symp., no.12, pp.1–6, 2006.
[39] Y. Sutani, T. Higashi, E. Tomita, S. Takahashi, and H. Nakatani, "A faster branch-and-bound algorithm for finding a maximum clique," Tech. Rep. IPSJ, 2006-AL-108, pp.79–86, 2006.
[40] S, Takahashi and E. Tomita, "Parallel computation for finding a maximum clique on shared memory computers," Tech. Rep. Univ. Electro-Commun., UEC-TR-CAS3, 2007.
[41] R.E. Tarjan and A.E. Trojanowski, "Finding a maximum independent set," SIAM J. Comput., vol.6, pp.537–546, 1977.
[42] E. Tomita and M. Yamada, "An algorithm for finding a maximum complete subgraph," Conference Records of the National Convention of IECE 1978, p.8, 1978.
[43] E. Tomita, Y. Kohata, and H. Takahashi, "A simple algorithm for finding a maximum clique," Tech. Rep. Univ. Electro-Commun., UEC-TR-C5(1), 1988.
[44] E. Tomita, S. Mitsuma, and H. Takahashi, "Two algorithms for finding a near-maximum clique," Tech. Rep. Univ. Electro-Commun., UEC-TR-C1, 1988.
[45] E. Tomita and T. Seki, "An efficient branch-and-bound algorithm for finding a maximum clique," Discrete Math. and Theoret. Comput. Sci., LNCS 2731, pp.278–289, 2003.
[46] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments (An invited paper in the Special Issue on COCOON 2004)," Theoret. Comput. Sci., vol.363, pp.28–42, 2006.
[47] E. Tomita and T. Kameda, "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.
[48] E. Tomita, Y. Sutani, T. Higashi, H. Takahashi, and M. Wakatsuki, "A simple and faster branch-and-bound algorithm for finding a maximum clique," WALCOM 2010, LNCS 5942, pp.191–203, 2010.
[49] E. Tomita, T. Akutsu, and T. Matsunaga, "Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics," in Biomedical Engineering, Trends in Electronics, Commun. and Software, ISBN: 978-953-307-475-7, InTech, ed. A.N. Laskovski, pp.625–640, 2011. Available from: http://www.intechopen.com/articles/show/title/efficient-algorithms-for-finding-maximum-and-maximal-cliques-effective-tools-for-bioinformatics
[50] M. Wakatsuki, S, Takahashi, and E. Tomita, "A parallelization of an algorithm for finding a maximum clique on shared memory computers," Tech. Rep. IPSJ, 2008-MPS-71, pp.17–20, 2008.
[51] D.R. Wood, "An algorithm for finding a maximum clique in a graph," Oper. Res. Lett., vol.21, pp.211–217, 1997.
[52] C. Yonemori, T. Matsunaga, J. Sekine, and E. Tomita, "A structural analysis of enterprise relationship using cliques," DBSJ J., vol.7, no.4, pp.55–60, March 2009.

Appendix: Clique Benchmark Results

Type of Machine: Pentium 4 3.6 GHz,
Main memory 2 Gbytes, Cache memory 1 Mbytes.
Compiler and flags used: gcc -O2.

In the second column of Table A· 1, we show our user time (T_1) required to solve each of the given *DIMACS benchmark instances:* `r100.5`, `r200.5`, `r300.5`, `r400.5`, and `r500.5` on our computer by the benchmark program dfmax [17], [18]. (*Correction* : Note that these values shown in the Appendix of [48] should be corrected as in Table A· 1. However, no other change of the values in [48] is necessary, since the other values in [48] are calculated based on the correct values. This correction is also noted on p.631 in Sect. 3.4 of [49].) For Östergård's user time for instances (T_2), Konc and Janežič's user time for instances (T_3), and Sewell's [32] user time for instances (T_4), by excluding the values of T_2/T_1, T_3/T_1, and T_4/T_1 for `r100.5` and `r200.5`, since these instances are extremely small, the average values of T_2/T_1, T_3/T_1, and T_4/T_1 are obtained as 4.48, 1.12, and 86.76, respectively. For Fahle's [10] user time (T_5), we obtained the average value of T_5/T_1 to be 2.03 through the dfmax running time for the common DIMACS benchmark graphs in [47]. Hence, Fahle's $\chi + DF$ running times for DIMACS benchmark graphs on our machine are calculated from [10]. Subsequently, for Régin's [30] user time (T_6), we obtained the average value of T_6/T_1 to be 1.35 by referring to the Fahle's $\chi + DF$ running time in [30] for the common DIMACS benchmark graphs which require more than 2,000 seconds on his machine.

Table A· 1 Each user time for instances [sec].

Graph	MCS	New [28]		MCDyn [21]		COCR [32]	
	Our	Östergård's		KJ's		Sewell's	
	T_1	T_2	T_2/T_1	T_3	T_3/T_1	T_4	T_4/T_1
`r100.5`	1.57×10^{-3}	0.01	6.37	0.00	0	0.14	89.17
`r200.5`	4.15×10^{-2}	0.23	5.54	0.04	0.96	3.64	87.71
`r300.5`	0.359	1.52	4.23	0.40	1.11	31.10	86.63
`r400.5`	2.21	10.05	4.55	2.48	1.12	191.98	86.87
`r500.5`	8.47	39.41	4.65	9.45	1.12	734.99	86.78

Etsuji Tomita received his B. Eng. and Dr. Eng. degrees in Electronics Engineering from Tokyo Institute of Technology, Japan, in 1966 and 1971, respectively. Then he was with the faculties of Tokyo Institute of Technology, and was appointed Associate Professor and subsequently Professor at the University of Electro-Communications (UEC Tokyo), Japan. He served as the founding Head of the Advanced Algorithms Research Laboratory at UEC Tokyo. He was also Professor at the Research and Development Initiative, Chuo University. He is presently Professor Emeritus at UEC Tokyo and is with the Advanced Algorithms Research Laboratory, UEC Tokyo, and is also Research Advisor of Japan Science and Technology Agency (JST) ERATO Minato Discrete Structure Manipulation System Project, and is with Tokyo Institute of Technology. His research interests include combinatorial optimization, algorithmic learning theory, and theory of automata and formal languages. He served as Director of Information Processing Society of Japan (IPSJ), Chair of Computer Science Domain of IPSJ, Program Committee Chair of ALT 2005, and Conference Chair of ICGI 2006. He is an Editor of ISRN Discrete Mathematics and some other journals. Dr. Tomita was given the Yonezawa Award of IECE, the Funai Information Technology Prize and was awarded Theoretical Computer Science Top Cited Article 2005–2010 for [46]. His book chapter [49] was downloaded over 8,000 times. He is a Fellow of IPSJ.

Yoichi Sutani received his B. Eng. and Master of Eng. degrees from the University of Electro-Communications in 2006 and 2008, respectively. He worked for algorithms for the maximum clique problem. Mr. Sutani was given IPSJ Yamashita SIG Research Award. He is with Sony Corporation from 2008.

Takanori Higashi received his B. of Eng. degree from the University of Electro-Communications in 2006. He worked for algorithms for the maximum clique problem. He is with Japan Systems Co., Ltd from 2006.

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 Graduate Course of Informatics, Graduate School of Informatics and Engineering at the University of Electro-Communications. His research interests include theory of automata and formal languages, computational learning theory, and combinatorial optimization problems. Dr. Wakatsuki is a member of the Information Processing Society of Japan and Japanese Society for Artificial Intelligence.