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

Etsuji Tomita**, Yoichi Sutani, Takanori Higashi,
Shinya Takahashi, and Mitsuo Wakatsuki

Advanced Algorithms Research Laboratory,
Department of Information and Communication Engineering
The University of Electro-Communications
Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, Japan
tomita@ice.uec.ac.jp

Abstract. This paper proposes new approximate coloring and other related techniques which markedly improve the run time of the branch-and-bound algorithm MCR (J. Global Optim., 37, 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. Consequently, it is shown by extensive computational experiments that MCS is remarkably faster than MCR and other existing algorithms. It is faster than the other algorithms by an order 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 graphs. MCS can be faster than MCR by a factor of more than 100,000 for some extremely dense random graphs.

1 Introduction

A *clique* is a subgraph 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 [3]. It is also difficult to obtain even a satisfactory approximate solution [12]. Nevertheless, many practical problems can be formulated as maximum clique problems (e.g., see [3], [6], [1], [5], [15], and others). Therefore, it is required to develop exact maximum-clique-finding algorithms that run very fast in practice.

* This research was supported in part by Grants-in-Aid for Scientific Research Nos. 16300001, 19500010, and 21300047 from the Ministry of Education, Culture, Sports, Science and Technology, Japan. It was also partially supported 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.

** Corresponding author. The author is also with the Research and Development Initiative, Chuo University, Kasuga 1-13-27, Bunkyo-ku, Tokyo 112-8551, Japan.

Md.S. Rahman and S. Fujita (Eds.): WALCOM 2010, LNCS 5942, pp. 191–203, 2010.

One standard approach to develop a fast algorithm is based on the branch-and-bound method, where the focus is on reducing the search space efficiently with *low overhead*. We developed a *simple* branch-and-bound algorithm that is referred to as MCR [23]; that was successful in reducing the search space with low overhead.

Here, *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. 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 set of vertices for approximate coloring. Following this ordered set of vertices, we present a new technique for reconstructing the adjacency matrix of a graph. The algorithm that is obtained by introducing these new techniques in MCR is named MCS. While MCS inherits the simplicity of MCR to a large extent, MCS is much more successful in reducing the search space quite efficiently. The main difference between the search spaces of MCR and MCS lies in the new approximate coloring together with the adjunct ordered set of vertices introduced in MCS. The resulting overhead in MCS is still low due to the *simplicity* of the newly introduced techniques. Consequently, extensive computational experiments have shown that MCS is remarkably faster than MCR and other algorithms. MCS is faster than other algorithms by an order 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 graphs.

MCR is only briefly described in Sect. 3 due to the page limitation, and the reader is advised to refer to [23] for further details.

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., $\Gamma(v) = \{w \in V | (v, w) \in E\}$. 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 an *induced* subgraph. An induced subgraph $G(Q)$ is said to be a *clique* if $(v, w) \in E$ for all $v, w \in Q \subseteq V$, with $v \neq w$. In this case, we may simply say that Q is a clique. 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 [23] 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* 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 that $\omega(R) \leq \mathrm{M}ax\{No[p]|p \in R\}$.

Hence, if $|Q| + \mathrm{M}ax\{No[p]|p \in R\} \leq |Q_{max}|$ holds, we need not continue the search for R.

After $Numbers$ ($Colors$) are assigned to all vertices in R, we sort the vertices in ascending order with respect to their $Numbers$. We refer to the numbering and sorting procedure as NUMBER-SORT [23]. In each step, select a vertex p in R, beginning from the last (right) vertex and ending at the first (left) vertex. As the result, a vertex with the *maximum* $Number$ is selected in constant time in each step. This is the reason why we sort the vertices in R with respect to their $Numbers$.

Let $\mathrm{M}ax\{No[r]|r \in R\} = maxno$ and $C_i = \{r \in R|No[r] = i\}$, where $i = 1, 2, \ldots, maxno$. 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 [24], 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 [23] is named *EXTENDED INITIAL SORT-NUMBER* to V.

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. 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 [10].

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 less than or equal to $|Q_{max}| - |Q|$. When we encounter such vertex p with $No[p] > |Q_{max}| - |Q|$, we attempt to change it's *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 $|C_{k_1}| = 1$.
2) If such q is found, then attempt to find *Number* k_2 such that no vertex in $\Gamma(q)$ has *Number* k_2.
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 *Number* k_2 is found, nothing is done.)

When the vertex q with *Number* k_2 is found, $No[p]$ is changed from No_p to k_1 ($\leq No_{th}$); thus, *it is no longer necessary to search from p*.

The exact procedure Re-NUMBER is shown in Fig. 1. To save time, we use it only when $No[p] = maxno$. The new approximate coloring is described in the first part of Fig. 2 under the heading {NUMBER}; it can be seen that Re-NUMBER follows the conventional greedy approximate coloring. The second part of Fig. 2, 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. 2, 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. 2, 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. 2) instead of the procedure NUMBER-SORT used in MCR [23] in order to make more effective use of the bounding condition.

```
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 |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_Nop := C_Nop − {p};
          C_k1 := (C_k1 − {q}) ∪ {p};
          C_k2 := C_k2 ∪ {q};
          return
        fi
      od
    fi
  od
end { of Re-NUMBER}
```

Fig. 1. Procedure Re-NUMBER

The time complexity of Re-NUMBER-SORT is $O(|R|^3)$, while that of NUMBER-SORT [23] is $O(|R|^2)$. Here, $|R|$ is the number of vertices of the concerned subgraph $G(R)$.

4.2 Adjunct Ordered Set of Vertices for Approximate Coloring

As noted in [7], [24], and [23], 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 [23]. 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 [9], [20]. However, the sorting of vertices is a computational burden and reduces the overall running time only for *dense* graphs [20]. The aim of the present study is to develop a faster algorithm whose use is not confined to any particular type of graphs. 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*

```
procedure Re-NUMBER-SORT(V_a, R, No)
begin
  {NUMBER}
    maxno := 0;
    C_1 := ∅;
    for i := 1 to |V_a| do
      { Conventional greedy approximate coloring }
      p := V_a[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};

      { - 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 = ∅ 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 ≠ 0 then
      R[i] := C_{k−1}[|C_{k−1}|];
      No[R[i]] := No_th
    fi
end { of Re-NUMBER-SORT }
```

Fig. 2. Procedure Re-NUMBER-SORT

order with respect to their degrees in the first stage [22]. We apply the procedure Re-NUMBER-SORT shown in Fig. 2 to the vertices in V_a, begining from the first (left) vertex and ending at the last (right) vertex. Thus, we can avoid the undesirable effect of Re-NUMBER.

```
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. 3. Algorithm MCS

As mentioned in Sect. 3.1, we select a vertex in the ordered set R for *searching*, beginning from the last (right) vertex and continuing up to the first (left) vertex, as shown in Fig. 3.

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. 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 in MCR is named MCS and is shown in Fig. 3.

4.5 Effectiveness of the Reduction of the Search Space

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. 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	6,608	15	9
r200.95	58–66	104,801	2,735	38	22
r200.98	90–103	2,357	4	589	155
r300.98	120	4.03×10^6	31,619	127	108
r500.994	263	$> 4.29 \times 10^6$	70	$> 61,286$	$> 256,410$
MANN_a45	345	2,952	225	13	11
p_hat500-3	50	138,300	7,923	18	12
p_hat700-3	62	3,733,665	88,168	42	29
san400_0.9_1	100	74	2	37	28
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	> 66

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 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 r$n.p$ ($n = 300, 500$ and $p = 0.98, 0.994$) are obtained from one random graph with n vertices and with edge probability p ($4.29 \times 10^9 = 2^{32}$). The number of branches is related to the size of the search space. 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 2 and 3 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.

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 [13], [14] 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 2. 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 2. The CPU times 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 [16] and COCR [18] are also listed for reference. The boldface entries indicate the fastest time in the row. In Table 2, 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 2, except for that with $[n = 200, p = 0.9]$. For this exceptional graph, COCR is approximately twice as fast as MCS. COCR is specially designed for solving the maximum clique problem for *dense* graphs. For the graphs with $p \geq 0.99$ in Table 2, MCS is faster than MCR by a factor of greater than 100,000.

Regarding dfmax, it was stated in [14] that "It ... may be hard to beat on sparser graphs, especially random ones." Prior to the development of MCQ [24], dfmax was widely recognized as the fastest maximum clique algorithm for sparse graphs, as stated in [8] and [16]. 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 2).

5.2 Results for DIMACS Benchmark Graphs

Table 3 lists the CPU times required by MCS and other algorithms to solve the DIMACS benchmark graphs [13], where the calibrated CPU times for New [16] and ILOG [17] 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

Table 2. CPU time[sec] for random graphs

Graph			dfmax	MCR		MCS	New	COCR
n	p	ω	[13]	[23]			[16]	[18]
	0.8	19-21	0.140	0.014	·	**0.008**	0.065	0.150
100	0.9	29-32	3.67	0.038	○	**0.013**	0.663	0.196
	0.95	39-48	23.736	0.011	○	**0.003**	0.196	
	0.98	56-68	26.5401	0.0012		**0.0009**		
	0.8	23	6.88	0.55	○	**0.23**		0.75
150	0.9	36-39	1058.96	5.26		**1.00**		1.16
	0.95	50-59	37,436.79	3.94	★	**0.35**		
	0.98	73-85	$> 10^5$	0.243	★○	**0.006**		
	0.8	24-27	192.7	12.3	·	**4.5**	147.3	8.7
200	0.9	40-44	$> 10^5$	647		74		○ **37**
	0.95	58-66	$> 10^5$	1,272	★○	**59**		
	0.98	90-103	$> 10^5$	30.9	★★	**0.2**		
	0.6	15-16	144.1	1.4		**1.0**	3.5	5.0
	0.7	19-21	26,236	23	·	**12**	121	
300	0.8	28-29	$> 10^5$	1,264	○	**394**		
	0.9	49		1,475,387	★○	**62,607**		
	0.98	120		284,534	★★	**2,623**		
	0.5	13-14	9.0	3.6		**2.8**	7.3	17.4
500	0.6	17	242	63	·	**40**	183	
	0.7	22-23	24,998	3,268	○	**1,539**		
	0.994	263	$> 1.5 \times 10^7$	$> 10^7$	★★★★★	**39**		
	0.4	12	33.3	16.1		**13.2**	23.2	
	0.5	15	1,107	395		**290**		
1,000	0.6	19-20	106,776	24,986	·	**15,317**		
	0.66	23		555,089	○	**275,964**		
	0.998	618		$> 10^7$	★★★★★	**46**		
2,000	0.9995	1,453		$> 10^7$	★★★★★	**61**		
	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, 100, 20, 10, 2, and 1.5 times faster than any of the others in the same row.

confirmed that MCS is almost always faster than MCR and the other algorithms in Table 3.

MCS is almost always considerably faster than χ+DF [8], COCR [18], MIPO [2], SQUEEZE [4], and Target [21] (see Table 4 in [23]). Although COCR is specially designed to efficiently find a maximum clique in dense graphs, MCS

Table 3. CPU time[sec] for DIMACS benchmark graphs (2)

Graphs				dfmax	MCR	MCS	New	ILOG
Name	n	*density*	ω	[13]	[23]		[16]	[17]
brock400_1	400	0.75	27	22,051	1,771	○ **693**		8,401
brock400_2	400	0.75	29	13,519	726	○ **297**		5,860
brock400_3	400	0.75	31	14,795	1,200	○ **468**		3,316
brock400_4	400	0.75	33	10,633	639	○ **248**		4,483
brock800_1	800	0.65	23	$> 10^5$	17,789	· **9,347**		$> 10,667$
brock800_2	800	0.65	24	$> 10^5$	16,048	· **8,368**		$> 10,667$
brock800_3	800	0.65	25	91,031	10,853	· **5,755**		$> 10,667$
brock800_4	800	0.65	26	78,737	7,539	· **3,997**		$> 10,667$
MANN_a27	378	0.990	126	$> 10^5$	2.5	○ **0.8**	$> 2,232$	13.7
MANN_a45	1,035	0.996	345	$> 10^5$	3,090	★ **281**		$> 10,667$
p_hat300-3	300	0.744	36	779.7	10.8	○ **2.5**		30.2
p_hat500-2	500	0.505	36	132.9	3.1	○ **0.7**	95.7	24.2
p_hat500-3	500	0.752	50	$> 10^5$	1,788	★ **150**		9,441
p_hat700-2	700	0.498	44	5,299.9	44.4	● **5.6**		189.5
p_hat700-3	700	0.748	62	$> 10^5$	68,187	★○ **2,392**		$> 10,667$
p_hat1000-2	1,000	0.489	46	$> 10^5$	2,434	★ **221**		12,478
p_hat1500-2	1,500	0.506	65	$> 10^5$	722,733	★○ **16,512**		$> 10,667$
san200_0.9_1	200	0.900	70	$> 10^5$	1.20	0.22	○ **0.06**	0.77
san200_0.9_2	200	0.900	60	$> 10^5$	4.2	○ **0.4**	1.0	1.9
san400_0.7_1	400	0.700	40	$> 10^5$	1.76	○ **0.54**	$> 2,232$	17.2
san400_0.7_2	400	0.700	30	$> 10^5$	0.33	○ **0.13**	112.97	50.0
san400_0.7_3	400	0.700	22	$> 10^5$	3.6	○ **1.4**		202.4
san400_0.9_1	400	0.900	100	$> 10^5$	3.4	★○ **0.1**		1,259.3
san1000	1,000	0.502	15	$> 10^5$	4.8	2.1	★○ **0.1**	76.1
sanr200_0.7	200	0.702	18	3.06	0.57	· **0.34**	3.15	3.2
sanr200_0.9	200	0.898	42	86,954	289	● **41**		111
sanr400_0.7	400	0.700	21	2,426	379	○ **181**		2,325
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		$> 10^7$	★● **151,597**		
gen400_p0.9_75	400	0.900	75		$> 10^7$	★○ **294,175**		
C250.9	250	0.899	44	$> 10^5$	44,214	★ **3,257**		

Entries marked ★★,★●, ★○, ★, ●, ○, and · are respectively at least 100, 50, 20, 10, 5, 2, 1.5 times faster than any of the others within the time limits in the same row.

is faster than COCR by 3.5 times for MANN_a27, whose density is very high (*density* = 0.990). Further, MCS is confirmed to be much faster than MC of Wood [26] and CP+SDP of Hoeve [11], as is evident in [17].

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*; hence, it runs remarkably faster than MCR and the other algorithms.

Owing to the page limitation of this paper, we cannot describe the details of the individual contribution of techniques in Sects. 4.1-4.3, but it is noted that effectiveness of MCS is established by the combination of all of these techniques. For example, a single introduction of the new approximate coloring (in Sect. 4.1) in MCR results in requiring more than 10^5 seconds to solve MANN_a45 [10]. A single introduction of the adjunct ordered set V_a (in Sect. 4.2) in MCR is almost always effective, but is not effective for MANN_a45 [22].

Our present techniques can be useful for generating large maximal cliques [25]. Some theoretical analysis of maximum-clique-finding algorithms is on the way based upon [25] and [19].

Acknowledgements. We thank T. Nakagawa for his contribution to the computational experiments. We express our sincere gratitude to E. Harley and the referees for their helpful detailed comments. Useful discussions and kind help by T. Akutsu and others are also acknowledged.

References

1. Bahadur, D.K.C., Tomita, E., Suzuki, J., Horimoto, K., Akutsu, T.: Protein threading with profiles and distance constraints using clique based algorithms. J. Bioinformatics and Computational Biology 4, 19–42 (2006)
2. Balas, E., Ceria, S., Cornuéjols, G., Pataki, G.: Polyhedral methods for the maximum clique problem. In: Johnson, Trick (eds.) [13], pp. 11–28 (1996)
3. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. In: Du, D.-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization, Supplement vol. A, pp. 1–74 (1999)
4. Bourjolly, J.-M., Gill, P., Laporte, G., Mercure, H.: An exact quadratic 0-1 algorithm for the stable set problem. In: Johnson, Trick (eds.) [13], pp. 53–73 (1996)
5. Brown, J.B., Bahadur, D.K.C., Tomita, E., Akutsu, T.: Multiple methods for protein side chain packing using maximum weight cliques. Genome Inform. 17, 3–12 (2006)
6. Butenko, S., Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics - invited review -. European J. Operational Research 173, 1–17 (2006)
7. Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem. Operations Research Letters 9, 375–382 (1990)
8. Fahle, T.: Simple and Fast: Improving a branch-and-bound algorithm for maximum clique. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 485–498. Springer, Heidelberg (2002)
9. Fujii, T., Tomita, E.: On efficient algorithms for finding a maximum clique. Technical Report of IECE, AL81-113, pp. 25–34 (1982)
10. Higashi, T., Tomita, E.: A more efficient algorithm for finding a maximum clique based on an improved approximate coloring. Technical Report of Univ. Electro-Commun, UEC-TR-CAS5-2006 (2006)
11. van Hoeve, W.J.: Exploiting semidefinite relaxations in costraint programming. Computers & Operations Research 33, 2787–2804 (2006)
12. Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Mathematica 182, 105–142 (1999)

13. Johnson, D.S., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiability, DIMACS Series in Discr. Math. and Theoret. Comput. Sci., vol. 26. American Math. Soc., Providence (1996)
14. `http://www.cs.sunysb.edu/~algorith/implement/dimacs/distrib/color/graph/form`
15. Matsunaga, T., Yonemori, C., Tomita, E., Muramatsu, M.: Clique-based data mining for related genes in a biomedical database. BMC Bioinformatics 10 (2009)
16. Östergård, P.R.J.: A fast algorithm for the maximum clique problem. Discrete Applied Math. 120, 197–207 (2002)
17. Régin, J.C.: Using constraint programming to solve the maximum clique problem. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 634–648. Springer, Heidelberg (2003)
18. Sewell, E.C.: A branch and bound algorithm for the stability number of a sparse graph. INFORMS J. Computing 10, 438–447 (1998)
19. Shindo, M., Tomita, E.: A simple algorithm for finding a maximum clique and its worst-case time complexity. Systems and Computers in Japan 21, 1–13 (1990)
20. Shindo, M., Tomita, E., Maruyama, Y.: An efficient algorithm for finding a maximum clique. Technical Report of IEC, CAS86-5, pp. 33–40 (1986)
21. Stix, V.: Target-oriented branch and bound method for global optimization. J. Global Optim. 26, 261–277 (2003)
22. Sutani, Y., Tomita, E.: Computational experiments and analyses of a more efficient algorithm for finding a maximum clique. Technical Report of IPSJ, 2005-MPS-57, pp. 45–48 (2005)
23. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Global Optim. 37, 95–111 (2007); J. Global Optim. 44, 311 (2009)
24. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731, pp. 278–289. Springer, Heidelberg (2003)
25. Tomita, E., Tanaka, A., Takahashi, H.: 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. 363, 28–42 (2006); The preliminary version appeared in Tomita, E., Tanaka, A. Takahashi, H.: The worst-case time complexity for generating all maximal cliques. In: Chwa, K.-Y., Munro, J.I.J.(eds.) COCOON 2004. LNCS, vol. 3106, pp.161–170. Springer, Heidelberg (2004)
26. Wood, D.R.: An algorithm for finding a maximum clique in a graph. Operations Research Letters 21, 211–217 (1997)

Appendix

Clique Benchmark Results

Type of Machine: Pentium 4 3.6 GHz, *Compiler and flags used:* gcc -O2.
Our user time (T_1) *for DIMACS benchmark instances:* `r100.5`, `r200.5`, `r300.5`, `r400.5`, and `r500.5` are 2.13×10^{-3}, 6.35×10^{-2}, 0.562, 3.48, and 13.3 seconds, respectively. From Östergård's [16] user time (T_2) and Sewell's [18] user time (T_3) for the same instances, we obtained the average values of T_2/T_1 and T_3/T_1 as 4.48 and 86.76, respectively, in the same way as in [23]. For Régin's [17] user time (T_5), we obtained the average value of T_5/T_1 to be 1.35 by referring to the $\chi + DF$ (Fahle's) [8] running time in [17].