The Worst-Case Time Complexity for Generating All Maximal Cliques*

(Extended Abstract)

Etsuji Tomita, Akira Tanaka**, and Haruhisa Takahashi

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. We present a depth-first search algorithm for generating all maximal cliques of an undirected graph, in which pruning methods are employed as in Bron and Kerbosch's algorithm. All maximal cliques generated are output in a tree-like form. Then we prove that its worst-case time complexity is $O(3^{n/3})$ for an n-vertex graph. This is optimal as a function of n, since there exist up to $3^{n/3}$ cliques in an n-vertex graph.

1 Introduction

In an undirected graph G, a *clique* is a complete subgraph of G in which any two vertices are adjacent. The set of vertices of a maximal clique of the complementary graph of G is a *maximal independent set* of G. Generating maximal cliques or maximal independent sets of a given graph is one of the fundamental problems in the theory of graphs and has many diverse applications, e.g., for clustering and for bioinformatics [10], [7]. A number of algorithms have been presented and evaluated experimentally or theoretically for this problem; see, e.g., [5], [14], [9], [6] and [4], where [4] is an elaborate survey paper on the maximum clique problem. Among them, Bron and Kerbosch [5] presented an algorithm for generating *all* maximal cliques of a graph together with some experimental results. Tsukiyama *et al.* [14] devised an algorithm for generating all maximal independent sets in a graph G in $O(nm\mu)$-time, where n, m, and μ are the numbers of vertices, edges, and maximal independent sets of G, respectively. Lawler *et al.* [9] generalized this result further. Chiba and Nishizeki [6] improved Tsukiyama *et al.*'s algorithm and gave a more efficient algorithm for listing all maximal cliques of G in $O(a(G)m\mu)$-time, where $a(G)$ is the arboricity of G with $a(G) \leq O(m^{1/2})$ for a connected graph G, and μ is the number of maximal cliques in G.

* This research has been supported in part by Grants-in-Aid for Scientific Research Nos. 13680435 and 16300001 from the Ministry of Education, Culture, Sports, Science and Technology, Japan, and Research Fund of the University of Electro-Communications. It is also given a grant by Funai Foundation for Information Technology.

** Presently with Toyota Techno Service Corporation, Imae 1-21, Hanamotochou, Toyota, Aichi 470–0334, Japan.

K.-Y. Chwa and J.I. Munro (Eds.): COCOON 2004, LNCS 3106, pp. 161–170, 2004.

We present here a depth-first search algorithm for generating all maximal cliques of an undirected graph, in which pruning methods are employed as in Bron and Kerbosch's algorithm [5]. All the cliques generated are output in a tree-like form. Then we prove that its worst-case running time complexity is $O(3^{n/3})$ for a graph with n vertices. This is the best one could hope for as a function of n, since there exist up to $3^{n/3}$ maximal cliques in a graph with n vertices as shown by Moon and Moser [11]. An earlier version of this paper appeared in [13].

2 Preliminaries

[1] Throughout this paper, we are concerned with a simple undirected *graph* $G = (V, E)$ with a finite set V of *vertices* and a finite set E of *unordered* pairs (v, w) of distinct vertices, called *edges*. A pair of vertices v and w are said to be *adjacent* if $(v, w) \in E$. We call $\overline{G} = (V, \overline{E})$ with $\overline{E} = \{(v, w) \in V \times V \mid v \neq w, \text{ and } (v, w) \notin E\}$ the *complementary* graph of G.

[2] For a vertex $v \in V$, let $\Gamma(v)$ be the set of all vertices which are adjacent to v in $G = (V, E)$, i.e., $\Gamma(v) = \{w \in V \mid (v, w) \in E\}$ ($\not\ni v$).

[3] For a subset $W \subseteq V$ of vertices, $G(W) = (W, E(W))$ with $E(W) = \{(v, w) \in W \times W \mid (v, w) \in E\}$ is called a *subgraph* of $G = (V, E)$ *induced* by W. For a set W of vertices, $\mid W \mid$ denotes the number of elements in W.

[4] Given a subset $Q \subseteq V$ of vertices, the induced subgraph $G(Q)$ is said to be *complete* if $(v, w) \in E$ for all $v, w \in Q$ with $v \neq w$. If this is the case, we may simply say that Q is a complete subgraph. A complete subgraph is also called a *clique*. If a clique is not a proper subgraph of another clique then it is called a *maximal* clique. A subset $W \subseteq V$ of vertices is said to be *independent* if $(v, w) \notin E$ for all $v, w \in W$. Here, $Q \subseteq V$ is a maximal clique of G if and only if Q is a maximal independent set of the complementary graph $\overline{G}$.

3 The Algorithm

We consider a depth-first search algorithm for generating all the cliques of a given graph $G = (V, E)$ $(V \neq \phi)$.

Here we introduce a global variable Q of a set of vertices which constitute a complete subgraph being found up to this time. Then we begin the algorithm by letting $Q := \phi$, and expand it step by step by applying a recursive procedure EXPAND to V and its succeeding induced subgraphs searching for larger and larger complete subgraphs until they reach maximal ones.

Let $Q = \{p_1, p_2, \ldots, p_d\}$ be found to be a complete subgraph at some stage, and consider a *subgraph* $G(SUBG)$ which is induced by a set of vertices

$$SUBG = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \cdots \cap \Gamma(p_d),$$

where $SUBG = V$ and $Q = \phi$ at the initial stage. Then apply the procedure EXPAND to $SUBG$ searching for larger complete subgraphs. If $SUBG = \phi$,

then Q is clearly a *maximal complete subgraph*, or, a *maximal clique*. Otherwise, $Q \cup \{q\}$ is a larger complete subgraph for every $q \in SUBG$. Then consider smaller subgraphs $G(SUBG_q)$ which are induced by new sets of vertices

$$SUBG_q = SUBG \cap \Gamma(q)$$

for all $q \in SUBG$, and apply recursively the same procedure EXPAND to $SUBG_q$ to find larger complete subgraphs containing $Q \cup \{q\}$.

Thus far we have shown only the basic framework of the algorithm for generating all maximal cliques (with possible duplications). This process can be represented by the following search forest, or the collection of search trees: The set of roots of the search forest is exactly the same as V of graph $G = (V, E)$. For each $q \in SUBG$, all vertices in $SUBG_q = SUBG \cap \Gamma(q)$ are sons of q. Thus, a set of vertices along a path from the root to any vertex of the search forest constitutes a complete subgraph, or a clique. We shall show an example of a search forest (with unnecessary subtrees deleted) later in Fig. 2 (b).

Now we proceed to describe two methods to prune unnecessary parts of the search forest. We regard the previously described set $SUBG$ $(\neq \phi)$ to be an *ordered* set of vertices, and we continue to generate maximal cliques from vertices in $SUBG$ step by step in this order .

First, let $FINI$ be the *first half* of $SUBG$ and suppose that we have already *finished* expanding search subtrees from every vertex $q' \in FINI \subseteq SUBG$ to generate *all* maximal cliques containig $Q \cup \{q'\}$, and that only the remaining vertex $q \in CAND = SUBG - FINI$ is a *candidate* for further expansion of the present complete subgraph Q to generate new larger cliques. That is, let

$$SUBG = FINI \cup CAND \quad (FINI \cap CAND = \phi),$$

with

$$FINI = \{q' \in SUBG \mid \textit{The procedure has finished generating all maximal cliques containing } Q \cup \{q'\}\}.$$

Consider the subgraph $G(SUBG_q)$ with $SUBG_q = SUBG \cap \Gamma(q)$, and let

$$SUBG_q = FINI_q \cup CAND_q \quad (FINI_q \cap CAND_q = \phi),$$

where

$$FINI_q = FINI \cap \Gamma(q), \text{ and } CAND_q = CAND \cap \Gamma(q).$$

Then only the vertices in the subgraph $G(CAND_q)$ can be candidates for expanding the complete subgraph $Q \cup \{q\}$ to find *new* larger cliques, since all the cliques containing $(Q \cup \{q\}) \cup \{r\}$ with $r \in FINI_q \subseteq FINI$ have already been generated for any r by application of the procedure EXPAND to $FINI$ as stated above. Thus, further expansion is to be considered only for vertices in $G(CAND_q)$ excluding ones in $FINI_q = SUBG_q - CAND_q$.

Secondly, given a certain vertex $u \in SUBG$, suppose that *all* the maximal cliques containing $Q \cup \{u\}$ have been generated. Then every *new* maximal clique

containing Q, but not $Q \cup \{u\}$, should contain at least one vertex $v \in SUBG - \Gamma(u)$. This is because if Q is expanded to a complete subgraph $R \subseteq (Q \cup S) \cap (SUBG - \{u\})$ with $S \subseteq \Gamma(u)$, then $R \cup \{u\}$ is a *larger* complete subgraph, and hence R is not *maximal*. Thus, any new maximal clique can be found by expanding Q to $Q \cup \{q\}$ such that $q \in SUBG - \Gamma(u)$, and by generating all the cliques containing $Q \cup \{q\}$. Therefore, if we have expanded a search subtree from $u \in SUBG$, then we should not expand any search subtree from $w \in SUBG \cap \Gamma(u)$.

Taking the previous pruning method also into consideration, the only search subtrees to be expanded are from vertices in $(SUBG - SUBG \cap \Gamma(u)) - FINI = CAND - \Gamma(u)$ $(\ni u)$. Here, in order to minimize $\mid CAND - \Gamma(u) \mid$, we choose such vertex $u \in SUBG$ to be the one which maximizes $\mid CAND \cap \Gamma(u) \mid$. In this way, the problem of generating all maximal cliques of $G(CAND)$ can be decomposed into $k = \mid CAND - \Gamma(u) \mid$ such subproblems; see **Lemma 1** (i) in Section 4.

Now we present an algorithm CLIQUES for generating all maximal cliques without duplications in Fig. 1.

If and only if Q is found to be a *maximal* clique at statement 2, we only print out a string of characters "*clique*," instead of Q itself at statement 3. Otherwise, it is impossible to achieve the worst-case running time of $O(3^{n/3})$ for an n-vertex graph, since printing out Q itself requires time proportional to the size of Q. Instead, in addition to statement 3, not only do we print out q followed by a *comma* at statement 7 every time q is picked out as a new element of a larger clique, but we also print out a string of characters "*back*," at statement 12 after q is moved from $CAND$ to $FINI$ at statement 11. We can easily obtain a tree representation of all the maximal cliques from the resulting sequence printed by statements 3, 7, and 12. Here, primed statements $0'$, $7'$, and $12'$ are only for the sake of explanation.

Example. Let us apply the above algorithm CLIQUES to a graph in Fig. 2 (a). Then the whole process is represented by a search forest in Fig. 2 (b), and we show the resulting printed sequence in Fig. 2 (c) *with appropriate indentations*. In Fig. 2 (b), each set of vertices surrounded by a flat circle represents $SUBG$ at that stage, in which vertex with $\triangle$ mark is in $FINI \subseteq SUBG$ at the beginning. Vertex u chosen at statement 4 is marked by $\boxdot$ or $\boxtimes$ depending on whether it is in $CAND$ or $FINI$, respectively. Other vertices in $CAND - \Gamma(u)$ are marked by $\circ$, while vertices in $CAND \cap \Gamma(u)$ are marked by $\bullet$. In conclusion, all the maximal cliques of G are $\{4, 6, 7, 8\}$, $\{4, 6, 5\}$, $\{4, 3, 8\}$, $\{1, 2, 9\}$, and $\{2, 3, 9\}$. □

Now given only the resulting printed sequence in Fig. 2 (c) without indentations, we can easily obtain essentially the same result as above by reconstructing from it a tree which represents a principal part of the previous search forest in Fig. 2 (b). Here, a dot "$\cdot$" $(\notin V)$ is introduced as a virtual root of the tree at the beginning. Then every time "q," is encountered in the sequence, we expand a downward edge whose end point is labeled by q. If and only if "q," is followed by "*clique*,", the set of all the vertices along the path from the root to the vertex q excluding the root ($\cdot$) represents a maximal clique. Every time "*back*," is

```
procedure CLIQUES(G)
              /* Graph G = (V, E) */
begin
 0′:/*   Q := φ                          */
      /* global variable Q is to constitute a clique */
 1 : EXPAND(V,V)
      procedure EXPAND(SUBG, CAND)
      begin
 2 :    if SUBG = φ
 3 :      then print ("clique,")
               /* to represent that Q is a maximal clique */
 4 :      else u := a vertex u in SUBG which maximizes | CAND ∩ Γ(u) |
               /* let EXT_u = CAND − Γ(u) */
 5 :           while CAND − Γ(u) ≠ φ
 6 :              do q := a vertex in (CAND − Γ(u))
 7 :                 print (q, ",")
                    /* to represent the next statement */
 7′: /*             Q := Q ∪ {q};        */
 8 :                SUBG_q := SUBG ∩ Γ(q);
 9 :                CAND_q := CAND ∩ Γ(q);
10:                 EXPAND(SUBG_q, CAND_q);
11:                 CAND := CAND − {q}    /* FINI := FINI ∪ {q} */
12:                 print ("back,")
                    /* to represent the next statement */
12′:/*              Q := Q − {q}         */
                 od
      fi
    end of EXPAND
end of CLIQUES
```

Fig. 1. Algorithm CLIQUES

encountered in the sequence, we go up the tree backward by one edge to find other maximal cliques. It is clear that this transformation can be done in time proportional to the length of the resulting sequence.

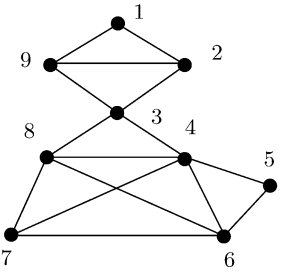
4 The Worst-Case Time Complexity

Given $G = (V, E)$ with $V \neq \phi$, we evaluate the worst-case running time of the previous algorithm CLIQUES(G) with the primed statements $0'$, $7'$, and $12'$ having been deleted. So, this is equivalent to evaluating that of EXPAND(V, V).

Now we begin by giving a few definitions.

[1] Let $T(n, m)$ be an upper bound on the worst-case running time of EXPAND ($SUBG$, $CAND$) when $\mid SUBG \mid = n$ and $\mid CAND \mid = m \quad (n \geq m \geq 1)$.

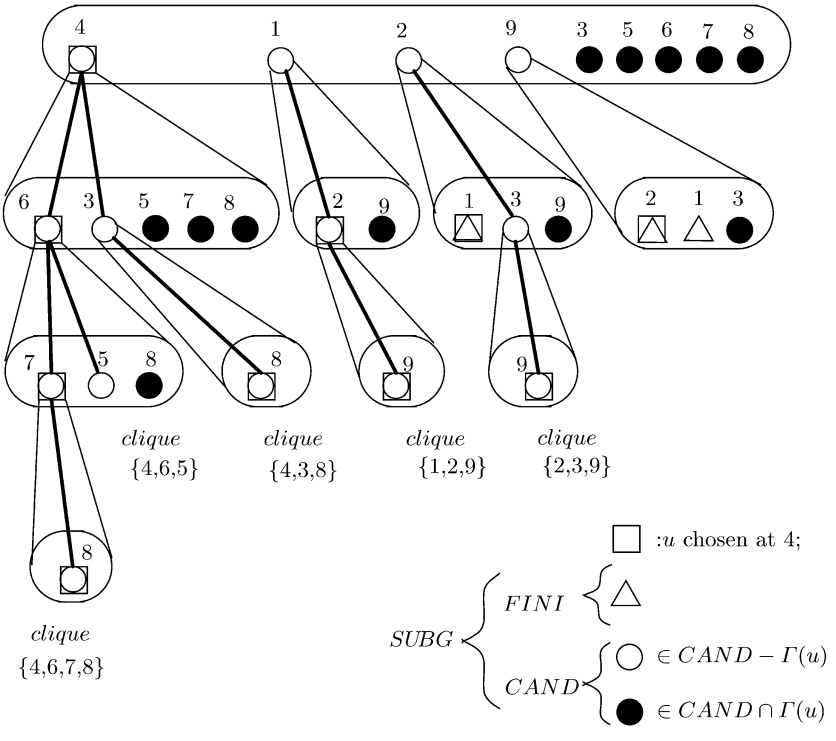
[2] Let $T_k(n, m)$ be an upper bound on the worst-case running time of EXPAND ($SUBG$, $CAND$) when $\mid SUBG \mid = n$, $\mid CAND \mid = m$, and $\mid EXT_u \mid = \mid CAND - \Gamma(u) \mid = k$ at the first entrance to statement 5.



(a) An input graph G

$4, 6, 7, 8, clique, back, back,$
$5, clique, back, back,$
$3, 8, clique, back, back, back,$
$1, 2, 9, clique, back, back, back,$
$2, 3, 9, clique, back, back, back,$
$9, back,$

(c) A resulting printed sequence



(b) A search forest for G

Fig. 2. An example

[3] Let us consider a nonrecursive procedure EXPAND$_0$($SUBG$, $CAND$) which is obtained from EXPAND($SUBG$, $CAND$) by replacing a recursive call 10: EXPAND ($SUBG_q$, $CAND_q$) by 10′:EXPAND(ϕ, ϕ). The running time of EXPAND$_0$($SUBG$, $CAND$) when $\mid SUBG \mid = n$ and $\mid CAND \mid = m$ can be made to be $O(n^2)$ (in which, selection of u at statement 4 can be done in $O(n^2)$), and so we assume that it is bounded above by the following quadratic formula

$$P(n) = p_1 n^2, \text{ where } p_1 > 0 \ . \qquad \square$$

From the above definitions, we have that

$$T(n,m) = \max_{1\le k\le m} \{T_k(n,m)\} \ . \tag{1}$$

The following Lemma is a key for evaluating $T(n,m)$.

Lemma 1. *Consider* EXPAND$(SUBG, CAND)$ *when* $\mid SUBG \mid = n$, $\mid CAND \mid = m$, $\mid EXT_u \mid = \mid CAND - \Gamma(u) \mid = k \neq 0$, *and* $\mid CAND \cap \Gamma(u) \mid = m - k$ *at the first entrance to statement 5. In what follows,* $CAND$ *stands exclusively for this initial value, though it is decreased one by one at statement 11 in the* **while** *loop. Let* $CAND - \Gamma(u) = \{v_1, v_2, \ldots, v_k\}$ *and the vertex at statement 6 be chosen in this order. Let*

$$SUBG_{v_i} = SUBG \cap \Gamma(v_i),$$

$$CAND_{v_i} = CAND \cap \Gamma(v_i), \ \textit{and}$$

$$CAND_i = (CAND - \{v_1, v_2, \ldots, v_{i-1}\}) \cap \Gamma(v_i).$$

Then the following hold.

(i) $T_k(\mid SUBG \mid, \mid CAND \mid) \le \sum_{i=1}^{k} T(\mid SUBG_{v_i} \mid, \mid CAND_i \mid) + P(n)$.
(ii) a) $\mid CAND_i \mid \le \mid CAND_{v_i} \mid \le m - k$.
b) $\mid SUBG_{v_i} \mid \le n - k \le n - 1$.

Proof. (i) It is obvious from **procedure** EXPAND$(SUBG, CAND)$ and the definition of $P(n)$.

(ii) a) $\mid CAND_i \mid \le \mid CAND_{v_i} \mid = \mid CAND \cap \Gamma(v_i) \mid \le \mid CAND \cap \Gamma(u) \mid = m - k$.
b) $\mid SUBG_{v_i} \mid = \mid SUBG \cap \Gamma(v_i) \mid$
$= \mid FINI \cap \Gamma(v_i) \mid + \mid CAND \cap \Gamma(v_i) \mid$,
with $\mid FINI \cap \Gamma(v_i) \mid \le \mid FINI \mid = n - m$ and $\mid CAND \cap \Gamma(v_i) \mid \le m - k$.
Here, $(n - m) + (m - k) = n - k \le n - 1$, since $k \ge 1$. **Q.E.D.**

Theorem 1. *For the upper bound* $T(n,m)$ *of the worst-case running time of* EXPAND$(SUBG, CAND)$ *with* $\mid SUBG \mid = n$ *and* $\mid CAND \mid = m$, *the following holds for all* $n \ge m \ge 1$ *:*

$$T(n,m) \le C3^{n/3} - Q(n) \equiv R(n), \tag{2}$$

where

$$Q(n) = q_1 n^2 + q_2 n + q_3,$$

with

$$q_1 = p_1/2 > 0, \quad q_2 = 9p_1/2 > 0, \quad q_3 = 27p_1/2 > 0,$$

and

$$C = \mathrm{Max}\{C_1, C_2, C_3\},$$

with $C_1 = 3q_2/\ln 3 = 27p_1/2 \ln 3$, $C_2 = 39p_1/(2 \cdot 3^{1/3})$, *and* C_3 *being the maximum over* n *of* $3(1 - 2 \cdot 3^{-2/3})^{-1} \cdot Q(n-3)/3^{n/3}$. *(Note that* $Q(n-3)/3^{n/3}$ *is bounded above, since it approaches* 0 *as* n *tends to infinity. Hence,* C_3 *is well-defined.)*

Here, $R(n) \equiv C3^{n/3} - Q(n)$ *is monotone increasing with* $R(n) \ge 0$ *for all integers* $n \ge 0$.

Proof. First, by differenciating a continuous function $R(x)$ with x being a real number and $C \geq C_1$, we can prove that $R(n)$ is monotone increasing for all integers $n \geq 0$. Furthermore, $R(1) = C3^{1/3} - Q(1) \geq C_2 3^{1/3} - 37p_1/2 = p_1$, since $C \geq C_2 = 39p_1/(2 \cdot 3^{1/3})$. So, $R(n) \geq p_1$ for all integers $n \geq 1$.

Now we prove that Eq.(2) holds by induction on n.

Basis. $n(= m) = 1$. We have $T(1,1) = P(1) = p_1$ by the definition of $P(n)$. So, Eq.(2) holds for $n = m = 1$, since $R(1) \geq p_1$.

Induction step. We assume that Eq.(2) holds for all integers n,m, $1 \leq m \leq n \leq N$, and prove that it also holds for $1 \leq m \leq n = N + 1$.

Consider EXPAND(*SUBG*,*CAND*) when $\mid SUBG \mid = n = N+1, \mid CAND \mid = m$ $(1 \leq m \leq n = N + 1)$, $\mid EXT_u \mid = \mid CAND - \Gamma(u) \mid = k \neq 0$ with $CAND - \Gamma(u) = \{v_1, v_2, \ldots, v_k\}$ at the first entrance to statement 5. Then just as in **Lemma 1** (i), we have

$$\begin{aligned} T_k(n,m) &= T_k(\mid SUBG \mid, \mid CAND \mid) \\ &\leq \sum_{i=1}^{k} T(\mid SUBG_{v_i} \mid, \mid CAND_i \mid) + P(n), \end{aligned}$$

where $\mid SUBG_{v_i} \mid \leq n - 1 = N$ by **Lemma 1** (ii)-b). Then by the induction hypothesis we have

$$\sum_{i=1}^{k} T(\mid SUBG_{v_i} \mid, \mid CAND_i \mid) \leq \sum_{i=1}^{k} R(\mid SUBG_{v_i} \mid).$$

Since $R(n)$ is monotone increasing and $\mid SUBG_{v_i} \mid \leq n - k$, we have

$$\sum_{i=1}^{k} R(\mid SUBG_{v_i} \mid) \leq kR(n-k).$$

Combining these inequalities followed by some calculations yields

$$\begin{aligned} T_k(n,m) &\leq C3^{n/3} - \{3Q(n-3) - P(n)\} \\ &= C3^{n/3} - Q(n) \quad (\text{ from the definition of } Q(n)). \end{aligned}$$

Substituting this inequality into Eq.(1), we have

$$T(n,m) \leq C3^{n/3} - Q(n).$$

Thus, Eq.(2) also holds for $n = N + 1 \geq m \geq 1$. Therefore, Eq.(2) has been induced to hold for *all* integers $n \geq m \geq 1$. Hence the result. **Q.E.D.**

In particular,

$$T(n,n) \leq C3^{n/3} - Q(n).$$

Therefore, we conclude that the worst-case running time of the algorithm CLIQUES(G) is $O(3^{n/3})$ for an n-vertex graph $G = (V, E)$.

Note here that if we output a list of all the individual maximal cliques, it takes $O(n3^{n/3})$ time in the worst case.

5 Concluding Remarks

We have given an algorithm for generating all maximal cliques in a graph of n-vertices and have proved that its worst-case time complexity is $O(3^{n/3})$ that is optimal as a function of n. It is regarded to be very hard to analyze theoretically the time complexity of our algorithm as a function of the number of maximal cliques in the graph.

The algorithm CLIQUES is very simple and is easy to be implemented. It is demonstrated by our computational experiments that CLIQUES runs much faster than the sophisticated algorithm CLIQUE [6] for several random graphs and Moon Moser graphs. In particular, CLIQUES is very much faster than CLIQUE for non-sparse graphs in which the number of maximal cliques is very large. In our experiments, for example, our algorithm CLIQUES runs more than 100 times faster than CLIQUE for random graphs of 180 vertices with edge density = 0.5, and for a Moon Moser graph of 45 vertices.

Our present depth-first search algorithm together with some heuristics yields an efficient algorithm for finding *a maximum* clique [12] that is experimentally demonstrated to outperform in general other existing such algorithms. It has been successfully applied to interesting problems in bioinformatics [1]–[3], the design of DNA and RNA sequences for bio-molecular computation [8], and others.

Note here that the algorithm in [12] for finding *a maximum* clique does not select a vertex with the maximum degree in each recursive call, which is in contrast to statement 4 in Fig. 1 of the present algorithm CLIQUES for generating *all* maximal cliques. This is because the algorithm[12] with a simple bounding rule by heuristics for finding *only one* maximum clique in each recuresive call takes less time than algorithms which are based on such a strategy as above for CLIQUES. In our present problem, however, we can not use such a bounding rule for finding *only one* maximum clique. In addition, *theoretical* time–complexity of an algorithm with heuristics is very hard to analyze. This is the reason why we employ here the strategies as in Bron and Kerbosch's algorithm.

Acknowledgment

The authors would like to acknowledge useful comments by J. Tarui, T. Akutsu, E. Harley, and the referees.

References

1. T. Akutsu, M. Hayashida, E. Tomita, J. Suzuki, and K. Horimoto: Protein threading with profiles and constraints, Proc. IEEE Symp. on Bioinformatics and Bioengineering (2004, to appear).
2. D. Bahadur K. C., 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 Informatics 13 (2002) 143–152.

3. D. Bahadur K. C., T. Akutsu, E. Tomita, and T. Seki: Protein side-chain packing: A maximum edge weight clique algorithmic approach, Proc. Asia-Pacific Bioinformatics Conf. (2004) 191–200.
4. I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo: The maximum clique problem. In: D.-Z. Du and P. M. Pardalos (eds.). Handbook of Combinatorial Optimization, Supplement vol. A, Kluwer Academic Publishers (1999) 1–74 .
5. C. Bron and J. Kerbosch: Algorithm 457, Finding all cliques of an undirected graph, Comm. ACM 16 (1973) 575–577.
6. N. Chiba and T. Nishizeki: Arboricity and subgraph listing algorithms, SIAM J. Comput. 14 (1985) 210–223.
7. M. Hattori, Y. Okuno, S. Goto, and M. Kanehisa: Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways. J. Am. Chem. Soc. 125 (2003) 11853–11865.
8. S. Kobayashi, T. Kondo, K. Okuda, and E. Tomita: Extracting globally structure free sequences by local structure freeness, Proc. DNA9 (2003) 206.
9. E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan: Generating all maximal independent sets, NP-hardness and polynomial-time algorithms, SIAM J. Comput. 9 (1980) 558–565.
10. S. Mohseni-Zadeh, A. Louis, P. Brézellec, and J.-L. Risler: PHYTOPROT: a database of clusters of plant proteins, Nucleic Acids Res. 32 (2004) D351-D353.
11. J. W. Moon and L. Moser: On cliques in graphs, Israel J. Math. 3 (1965) 23–28.
12. E. Tomita and T. Seki: An efficient branch-and-bound algorithm for finding a maximum clique, DMTCS 2003, Lec. Notes in Comput. Sci. 2731 (2003) 278–289.
13. E. Tomita, A. Tanaka, and H. Takahashi: An optimal algorithm for finding all the cliques, Technical Report of IPSJ, 1989-AL-12 (1989) 91–98.
14. S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa: A new algorithm for generating all the maximal independent sets, SIAM J. Comput. 6 (1977) 505–517.