Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 363 (2006) 28 – 42

Theoretical Computer Science

www.elsevier.com/locate/tcs

The worst-case time complexity for generating all maximal cliques and computational experiments

Etsuji Tomita[a,*], Akira Tanaka[a,b], Haruhisa Takahashi[a]

[a]The University of Electro-Communications, Department of Information and Communication Engineering, Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, Japan

[b]Toyota Techno Service Corporation, Imae 1-21, Hanamotocho, Toyota, Aichi 470–0334, Japan

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 the Bron–Kerbosch algorithm. All the maximal cliques generated are output in a tree-like form. Subsequently, 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 $3^{n/3}$ maximal cliques in an n-vertex graph. The algorithm is also demonstrated to run very fast in practice by computational experiments.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Maximal cliques; Enumeration; Worst-case time complexity; Computational experiments

1. Introduction

In an undirected graph G, a *clique* is a complete subgraph of G, i.e., a subgraph 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 such a generation has many diverse applications, e.g., in clustering and in bioinformatics [15,16,8]. A number of algorithms have been presented and evaluated experimentally or theoretically for this problem [5–7,9,26,13,19,12]. In particular, Pardalas and Xue [19], and Bomze et al. [5] conducted surveys on the maximum clique problem. Bron and Kerbosch [6] presented a depth-first search algorithm for generating *all* the maximal cliques of a graph. They showed experimentally that the computing time per clique is almost independent of the graph size for *random* graphs and that the total computing time is proportional to $(3.14)^{n/3}$ for *Moon–Moser* graphs [17] of n vertices. Furthermore, Johnston [9] and Koch [12] presented some variations of the Bron–Kerbosch algorithm together with a considerable number of computational experiments. No results, however, were ever published on the theoretical time complexity for these algorithms. On the other hand, Tsukiyama et al. [26] devised an algorithm for generating all the maximal independent sets in a graph G in $O(nm\mu)$-time, where n, m, and μ are the number of vertices, edges, and maximal independent sets of G, respectively. Lawler et al. [13] generalized this result further. Chiba and Nishizeki [7] improved the algorithm of Tsukiyama

* Corresponding author.
E-mail addresses: tomita@ice.uec.ac.jp (E. Tomita), akirat@mc.neweb.ne.jp (A. Tanaka), takahasi@ice.uec.ac.jp (H. Takahashi).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.06.015

et al. and presented a more efficient algorithm for listing all the maximal cliques of G in $\mathrm{O}(a(G)m\mu)$-time, where $a(G)$ is the arboricity of G with $a(G) \leqslant \mathrm{O}(m^{1/2})$ for a connected graph G and μ is the number of maximal cliques in G.

Recently, Makino and Uno [14] presented new algorithms, which are based on the algorithm of Tsukiyama et al. [26]. One of their algorithms enumerates all the maximal cliques of G in $\mathrm{O}(\Delta^4\mu)$-time, where Δ is the maximal degree of G (Theorem 2 of [14]). They showed by using computational experiments that the algorithms is considerably faster than that of Tsukiyama et al. [26] for sparse graphs.

We present here a depth-first search algorithm for generating all the maximal cliques of an undirected graph, in which pruning methods are employed as in the Bron–Kerbosch algorithm [6]. All the maximal cliques generated are output in a tree-like form. Subsequently, we prove that its worst-case running time complexity is $\mathrm{O}(3^{n/3})$ 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 [17]. Our algorithm differs from that of Bron and Kerbosch in the format of the printed maximal cliques. The difference is important theoretically, since it is essential to the establishment of the *optimal* worst-case time complexity. It is also very important practically, since it saves space in the output file. The Bron–Kerbosch algorithm and other algorithms that yield the maximal cliques in a straightforward manner may require huge amounts of memory space compared to our algorithm.

The time complexity of our algorithm is of special interest when it is compared with the results of Tarjan and Trojanowski [21] and Robson [20]. The former presented an $\mathrm{O}(2^{n/3})$-time algorithm for finding only *one maximum* independent set—one independent set with the largest number of vertices—and the latter showed an $\mathrm{O}(2^{0.276n})$-time algorithm (using exponential space) for the same problem.

The time complexity of our algorithm is expressed as a function of n, the number of vertices, while the time complexities of the algorithms developed by Tsukiyama et al. and their successors are expressed as a function of μ, the number of maximal cliques (or maximal independent sets); therefore a direct theoretical comparison of these algorithms is difficult. We have performed extensive computational experiments and demonstrated that our algorithm is much faster in practice than those of Tsukiyama et al. and their successors.

Earlier versions of this paper appeared in Tomita et al. [23–25]; in particular, the version of Tomita et al. [23] was reviewed by Pardalos and Xue [19] and Bomze et al. [5] and received considerable attention.

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 $(v, w) \in E$.

[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., $\Gamma(v) = \{w \in V \mid (v, w) \in E\}$ $(\not\ni v)$.

[3] For the 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, $|W|$ denotes the number of elements in W.

[4] Given the 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 $v \neq w$. In this case, we may simply state 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.

3. The algorithm

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

Here, we introduce a global variable Q of a set of vertices that constitutes a complete subgraph found up to this time. The algorithm begins by letting Q be an empty set, and expands Q step by step by applying a recursive procedure EXPAND to V and its succeeding induced subgraphs to search for larger and larger complete subgraphs until they reach maximal ones.

Let $Q = \{p_1, p_2, \ldots, p_d\}$ be a complete subgraph found at some stage, and consider the set of vertices

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

where $SUBG = V$ and $Q = \emptyset$ at the initial stage. Apply the procedure EXPAND to $SUBG$ to search for larger complete subgraphs. If $SUBG = \emptyset$ then Q is clearly a $\underline{maximal}$ *complete subgraph*, or a $\underline{maximal}$ *clique*. Otherwise, $Q \cup \{q\}$ is a larger complete subgraph for every $q \in SUBG$. Now, consider the smaller subgraphs $G(SUBG_q)$ that are induced by new sets of vertices

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

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

Thus far, we have described only a well-known basic framework of the algorithm for generating all the maximal cliques (with possible duplication). 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 the vertices in $SUBG_q$ (defined above) are children 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 give an example of a search forest (with unnecessary subtrees deleted) later in Fig. 3(b).

Now we describe two methods to prune unnecessary parts of the search forest, which are found to be the same as in the Bron–Kerbosch algorithm [6]. We regard the previously described set $SUBG$ ($\neq \emptyset$) as an *ordered* set of vertices, and we continue to generate maximal cliques from the vertices in $SUBG$ stepwise in this order.

First, let $FINI$ be a subset of vertices of $SUBG$ that have already been processed by the algorithm ($FINI$ is short for "*finished*"). We then denote the set of remaining candidates for expansion by $CAND$: $CAND = SUBG - FINI$. Hence, we have

$$SUBG = FINI \cup CAND \quad (FINI \cap CAND = \emptyset).$$

$FINI = \emptyset$ at the beginning. Consider the subgraph $G(SUBG_q)$ with $SUBG_q$ as defined above, and let

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

where

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

Following this, only the vertices in $CAND_q$ 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 (see Fig. 1).

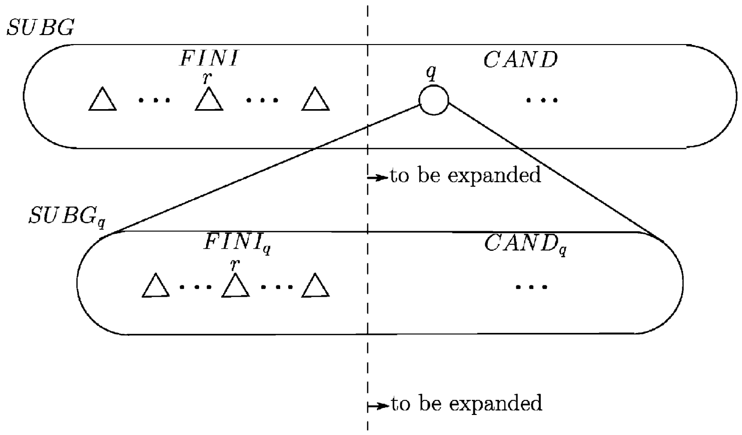
Secondly, given a certain vertex $u \in SUBG$, consider that *all* the maximal cliques containing $Q \cup \{u\}$ have been generated. Then, every *new* maximal clique containing Q, but not $Q \cup \{u\}$, must contain at least one vertex $q \in SUBG - \Gamma(u)$. This is because if Q is expanded to a complete subgraph $R = (Q \cup S) \cap (SUBG - \{u\})$ with $S \subseteq SUBG \cap \Gamma(u)$, 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 subsequently generating all the cliques containing $Q \cup \{q\}$.

Taking the previously described pruning method also into consideration, the only search subtrees to be expanded are from the vertices in $(SUBG - SUBG \cap \Gamma(u)) - FINI = CAND - \Gamma(u)$. Here, in order to minimize $|CAND - \Gamma(u)|$, we choose a vertex $u \in SUBG$ as the one that maximizes $|CAND \cap \Gamma(u)|$. This is essential for the proof of Lemma 2(ii), and consequently the main theorem, i.e., Theorem 3. In this manner, the problem of generating all maximal cliques of $G(CAND)$ can be decomposed into $k = |CAND - \Gamma(u)|$ such subproblems; see Lemma 2(i) in Section 4.

We present an algorithm CLIQUES for generating all the maximal cliques without duplication in Fig. 2. Note here that some branches may not generate a maximal clique, as in the case where $SUBG$ is not empty but $CAND - \Gamma(u)$ is empty.

If Q is a *maximal* clique that is found at statement 2, then the algorithm only prints 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 the printing of Q itself requires time proportional to the size of Q. Instead, in addition to printing

$Q = \{p_1, p_2, \ldots, p_d\}$: complete subgraph found up to this time
$SUBG = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \ldots \cap \Gamma(p_d) = FINI \cup CAND$.
$Q \cup \{q\}$: larger complete subgraph



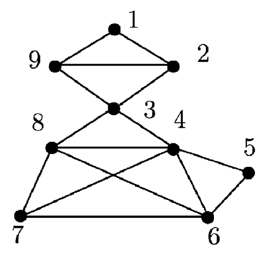
$SUBG_q = SUBG \cap \Gamma(q) = FINI_q \cup CAND_q$
$FINI_q = FINI \cap \Gamma(q)$
$CAND_q = CAND \cap \Gamma(q)$

Fig. 1. An illustration of the procedure EXPAND.

```
procedure  CLIQUES($G$)
             /* Graph $G = (V, E)$ */
begin
0':/*    $Q := \emptyset$;                    */
       /* global variable $Q$ is to constitute a clique */
1 : EXPAND($V$,$V$)
end of CLIQUES

       procedure EXPAND($SUBG$, $CAND$)
       begin
2 :      if $SUBG = \emptyset$
3 :        then print ("clique,")
               /* to represent that $Q$ is a maximal clique */
4 :        else $u$ := a vertex $u$ in $SUBG$ that maximizes $| CAND \cap \Gamma(u) |$;
               /* let $EXT_u = CAND - \Gamma(u)$; */
               /*     $FINI := \emptyset$;          */
5 :            while $CAND - \Gamma(u) \neq \emptyset$
6 :              do $q$ := a vertex in $(CAND - \Gamma(u))$;
7 :                 print ($q$, ",");
                    /* to represent the next statement */
7': /*              $Q := Q \cup \{q\}$;       */
8 :                 $SUBG_q := SUBG \cap \Gamma(q)$;
9 :                 $CAND_q := CAND \cap \Gamma(q)$;
10:                 EXPAND($SUBG_q$, $CAND_q$);
11:                 $CAND := CAND - \{q\}$;   /* $FINI := FINI \cup \{q\}$; */
12:                 print ("back,");
                    /* to represent the next statement */
12':/*              $Q := Q - \{q\}$         */
                 od
         fi
       end of EXPAND
```

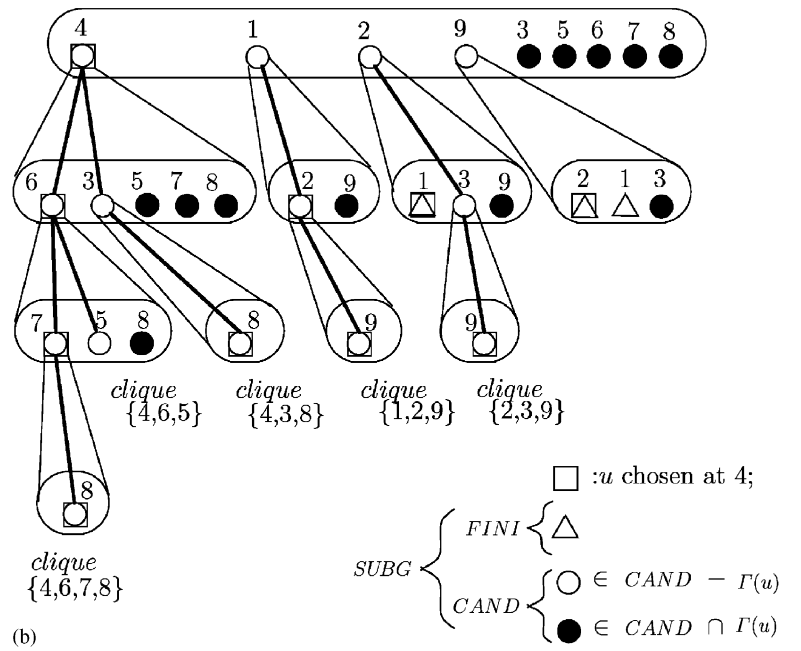
Fig. 2. Algorithm CLIQUES.



(a)

$4, 6, 7, 8, \mathit{clique}, \mathit{back}, \mathit{back},$
$\quad 5, \mathit{clique}, \mathit{back}, \mathit{back},$
$\quad 3, 8, \mathit{clique}, \mathit{back}, \mathit{back}, \mathit{back},$
$1, 2, 9, \mathit{clique}, \mathit{back}, \mathit{back}, \mathit{back},$
$2, 3, 9, \mathit{clique}, \mathit{back}, \mathit{back}, \mathit{back},$
$9, \mathit{back},$

(c)



(b)

Fig. 3. An example (a) An input graph G; (b) A search forest for G; and (c) A resulting printed sequence.

"clique" at statement 3, we print q followed by a *comma* at statement 7 every time q is selected as a new element of a larger clique; further we 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 sequence printed by statements 3, 7, and 12. In Fig. 2, the primed statements ($0'$, $7'$, and $12'$) are solely for the sake of explanation.

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

Given only the resulting printed sequence in Fig. 3(c) without indentations, we can easily obtain essentially the same result as above by reconstructing from it a tree that represents a principal component of the previous search forest in Fig. 3(b). Here, a dot $\cdot$ ($\notin V$) is introduced as a virtual root of the tree. Subsequently, 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 encountered in the sequence, we go up the tree backward by one edge to find other maximal

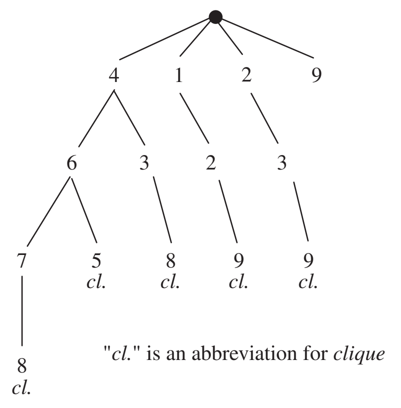


Fig. 4. A tree representation of the result of the example in Fig. 3.

cliques. See Fig. 4 for a tree representation of the result of the Example. It is clear that this transformation can be done in a time that is proportional to the length of the resulting sequence.

We conclude this section by the following theorem.

Theorem 1 (*Correctness of the algorithm CLIQUES*). *Given a graph $G = (V, E)$ $(V \neq \emptyset)$, the algorithm CLIQUES generates all and only maximal cliques without duplication.*

Proof. See Appendix A.1.

4. The worst-case time complexity

Given $G = (V, E)$ with $V \neq \emptyset$, we evaluate the worst-case running time of the algorithm *CLIQUES*(G). This is equivalent to evaluating the worst-case running time of EXPAND(V, V).

We begin with the following definitions.

[1] Let $T(n)$ be the worst-case running time of EXPAND(*SUBG*, *CAND*) when $|SUBG| = n \quad (n \geqslant 1)$.

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

[3] Let us consider a nonrecursive procedure EXPAND$_0$(*SUBG*, *CAND*) that is obtained from EXPAND(*SUBG*, *CAND*) by replacing a recursive call 10: EXPAND$(SUBG_q, CAND_q)$ by 10′:EXPAND$(\emptyset, \emptyset)$. The running time of EXPAND$_0$(*SUBG*, *CAND*) when $|SUBG| = n$ 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 the running time of EXPAND$_0$(*SUBG*, *CAND*) is bounded above by the following quadratic formula

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

From the above definitions, we have

$$T(n) = \max_k \{T_k(n)\}, \tag{1}$$

where $1 \leqslant k \leqslant |CAND|$.

The following lemma is a key for evaluating $T(n)$.

Lemma 2. *Consider* EXPAND(*SUBG*, *CAND*) *when* $|SUBG| = n$, $|EXT_u| = |CAND - \Gamma(u)| = k \neq 0$, *and* $|CAND \cap \Gamma(u)| = |CAND| - k$ *at the first entrance to statement* 5. *In what follows*, *CAND stands exclusively for this initial value*, *although it is repeatedly decreased at statement* 11 *in the* while *loop. Let* $CAND - \Gamma(u) = \{v_1, v_2, \ldots, v_k\}$

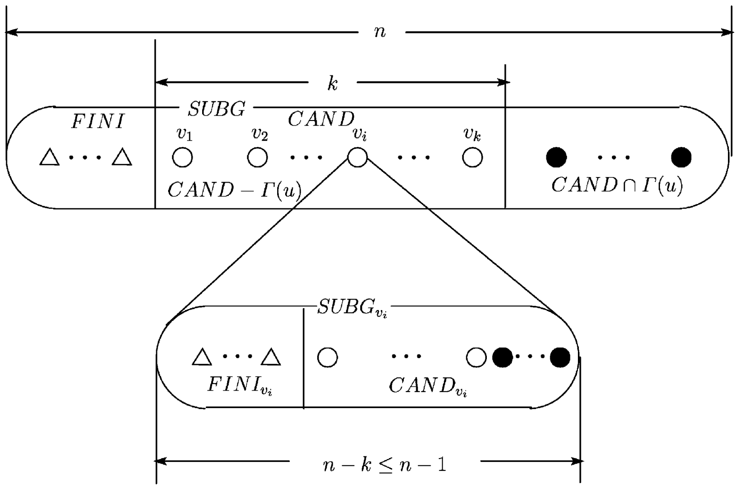


Fig. 5. An illustration for Lemma 2.

and let the vertex at statement 6 *be chosen in this order. Let*

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

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

(*See Fig*. 5.)

Then the following hold.

(i) $T_k(|SUBG|) \leqslant \sum_{i=1}^{k} T(|SUBG_{v_i}|) + P(n)$.

(ii) $|SUBG_{v_i}| \leqslant n - k \leqslant n - 1$.

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

(ii) Since the vertex u in *SUBG* is chosen so that it maximizes $|CAND \cap \Gamma(u)|$ we have $|CAND \cap \Gamma(v_i)| \leqslant |CAND \cap \Gamma(u)| = |CAND| - k$. Hence, $|SUBG_{v_i}| = |SUBG \cap \Gamma(v_i)| = |FINI \cap \Gamma(v_i)| + |CAND \cap \Gamma(v_i)| \leqslant |FINI| + |CAND \cap \Gamma(v_i)| \leqslant (n - |CAND|) + (|CAND| - k) = n - k \leqslant n - 1$, since $k \geqslant 1$. □

Theorem 3. *For all* $n \geqslant 1$

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

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

$$C = \max\{C_1, C_2, C_3\},$$

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* $R(n) \geqslant 0$ *for all integers* $n \geqslant 0$.

Proof. By considering a *continuous* function $R(x)$ with x being a real variable and the fact that $C \geqslant C_1$, $R(x)$ can be easily proved to be monotone increasing for $x \geqslant 0$ (Lemma A.2 in Appendix). Hence, $R(n)$ is monotone increasing for all integers $n \geqslant 0$. Furthermore, $R(1) = C3^{1/3} - Q(1) \geqslant C_2 3^{1/3} - 37p_1/2 = p_1$, since $C \geqslant C_2 = 39p_1/(2 \cdot 3^{1/3})$. Therefore, $R(n) \geqslant p_1 > 0$ for all integers $n \geqslant 1$.

Now, we prove that inequality (2) holds by induction on *n*.

Basis: $n = 1$. We have $T(1) = P(1) = p_1$ by the definition of $P(n)$. Therefore, inequality (2) holds for $n = 1$, since $R(1) \geqslant p_1$.

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

Consider EXPAND(*SUBG*, *CAND*) when $|SUBG| = n = N + 1$, $|EXT_u| = |CAND - \Gamma(u)| = k \neq 0$ with $CAND - \Gamma(u) = \{v_1, v_2, \ldots, v_k\}$ at the first entrance to statement 5. Then, just as in Lemma 2(i), we have

$$T_k(n) = T_k(|SUBG|) \leqslant \sum_{i=1}^{k} T(|SUBG_{v_i}|) + P(n),$$

where $|SUBG_{v_i}| \leqslant n - 1 = N$ by Lemma 2(ii). Then, by the induction hypothesis we have

$$\sum_{i=1}^{k} T(|SUBG_{v_i}|) \leqslant \sum_{i=1}^{k} R(|SUBG_{v_i}|).$$

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

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

Combining these inequalities yields

$$\begin{aligned} T_k(n) &\leqslant kR(n-k) + P(n) \\ &\equiv kC3^{(n-k)/3} - kQ(n-k) + P(n) \\ &= k3^{-k/3} \cdot C3^{n/3} - \{kQ(n-k) - P(n)\}. \end{aligned} \tag{3}$$

In the case $k = 3$, we have

$$T_3(n) \leqslant C3^{n/3} - \{3Q(n-3) - P(n)\}.$$

Now, consider the case where $k \neq 3$ (with $k \geqslant 1$). We shall show that

$$k3^{-k/3} \cdot C3^{n/3} - \{kQ(n-k) - P(n)\} \leqslant C3^{n/3} - \{3Q(n-3) - P(n)\} \tag{4}$$

for all integers $n \geqslant 1$, provided that $C \geqslant C_3$. Modifying inequality (4), the problem is equivalent to proving the following inequality.

$$\frac{3Q(n-3) - kQ(n-k)}{(1 - k3^{-k/3}) \cdot 3^{n/3}} \leqslant C \quad \text{where } k \neq 3. \tag{5}$$

Here, $kQ(n-k) \geqslant 0$ for $1 \leqslant k \leqslant n$. Therefore, for the left-hand side of inequality (5), we have

$$\frac{3Q(n-3) - kQ(n-k)}{(1 - k3^{-k/3}) \cdot 3^{n/3}} \leqslant \frac{3Q(n-3)}{(1 - 2 \cdot 3^{-2/3}) \cdot 3^{n/3}} \leqslant C_3. \tag{6}$$

Here, the first inequality holds since $k3^{-k/3} \leqslant 2 \cdot 3^{-2/3}$ (Lemma A.3 in Appendix) and the second inequality comes from the definition of C_3.

Now from the fact that $C \geqslant C_3$, inequality (4) holds for all integers $n \geqslant 1$ and $1 \leqslant k \leqslant n$.

Combining inequalities (3) and (4) yields

$$T_k(n) \leqslant C3^{n/3} - \{3Q(n-3) - P(n)\} = C3^{n/3} - Q(n) \quad \text{(from the definition of } Q(n)).$$

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

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

Thus, inequality (2) also holds for $n = N + 1$. Therefore, inequality (2) holds for *all* integers $n \geqslant 1$. Hence the result. □

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

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

5. Computational experiments

In addition to the preceding theoretical analysis of the algorithm CLIQUES, we have implemented CLIQUES in the programming language C and have performed computational experiments to evaluate it in practice [18]. The computer used had a Pentium4 2.20 GHz CPU with 2 GB main memory and a Linux operating system. The compiler and flags used are gcc-O2. These are the same as those in [22]. (So, Clique Benchmark Results in Appendix of [22] and dfmax CPU time for DIMACS benchmark graphs in Table 3 of [22] can be effectively used for calibrating CPU times in different computer environments.)

For comparison, we have also implemented and evaluated the algorithm CLIQUE of Chiba and Nishizeki [7], the algorithms ALLMAXCLIQUES (AMC for short) and ALLMAXCLIQUES* (AMC* for short) of Makino and Uno [14] in the same manner as above. In these experiments, the algorithms other than CLIQUES are modified to yield only the number of maximal cliques in order to exclude the requirement of a huge amount of memory space for listing all the maximal cliques.

It is to be noted that the working of AMC* of [14] depends on the value of Δ^* for the input graph G such that only some number (θ) of vertices in G have a degree larger than Δ^*. To evaluate the overall performance of AMC* that is independent of the input graphs, we set $\Delta^* = n/100$ for any input graph of n vertices. This is because it is not necessarily easy to choose an appropriate value of Δ^* for an unknown graph. In addition, we have confirmed in the following experiments that AMC* with $\Delta^* = n/100$ is much faster than the basic algorithm of Makino and Uno that is based on Theorem 2 of [14]; that was used in their computational experiments in Section 8 of [14]. Moreover, this AMC* is faster than AMC*s with other settings of Δ^* for a wide variety of graphs.

First, a random graph is generated for each pair of n (the number of vertices) and p (edge probability) in Table 1 so that every pair of vertices has an edge with a probability p. The third column titled "♯cliques" in Table 1 shows the number of maximal cliques of each random graph generated as discussed above. The CPU time in seconds needed to yield the number of maximal cliques for each random graph using CLIQUE [7], AMC, and AMC* [14] are listed in the fourth, fifth, and sixth columns, respectively (24 h = 86, 400 sec). The last but one column presents the CPU time using CLIQUES for each random graph. The bold-faced entries are the fastest in a given row. The final column titled "/cliques" shows the average CPU time in seconds for generating 10^6 maximal cliques, i.e., (CPU time/♯cliques) $\times 10^6$, for each graph.

Table 2 shows the corresponding CPU time using CLIQUE, AMC, AMC*, and CLIQUES for Moon–Moser graphs [17] and DIMACS benchmark graphs [10], where M–M–n stands for Moon–Moser graphs of n vertices, and density stands for edge density of the graph, i.e., (the number of edges in the graph)/$\{n(n-1)/2\}$ for an n-vertex graph.

To compare our experimental results with those of [14], we have also examined sparse *locally* random graphs proposed in [14]. Their locally random graphs are generated such that edges exist randomly only in certain limited

Table 1
CPU time (sec) for random graphs

Graphs			CLIQUE	AMC	AMC*	CLIQUES	/cliques
n	p	♯cliques	[7]	[14]	[14]		
	0.6	65,216	10.38	2.81	1.60	**0.088**	1.30
100	0.7	415,412	107.13	16.64	9.52	**0.54**	1.21
	0.8	5,467,664	2,035.18	179.51	112.06	**6.63**	1.05
	0.9	313,069,088	> 24 h	7,434.26	6,533.98	**330.19**	4.41
	0.1	3,882	0.62	0.26	0.067	**0.0041**	1.06
	0.2	18,737	5.33	3.26	1.04	**0.022**	1.17
	0.3	96,298	41.28	26.77	9.55	**0.14**	1.45
300	0.4	559,838	364.89	197.20	78.26	**0.88**	1.57
	0.5	4,874,385	5,645.05	1,759.61	789.23	**8.54**	1.75
	0.6	132,240,024	> 24 h	24,104.49	12,937.56	**140.75**	1.06
	0.7	3,356,452,714	> 24 h	> 24 h	> 24 h	**6,279.51**	1.87
	0.1	15,252	6.04	3.13	0.69	**0.018**	1.18
500	0.2	99,259	72.84	48.60	14.17	**0.13**	1.31
	0.3	728,567	812.10	814.98	196.63	**1.19**	1.63
	0.5	97,419,729	> 24 h	> 24 h	42,612.27	**208.16**	2.14
	0.1	37,563	29.91	21.86	3.15	**0.051**	1.36
700	0.2	325,479	485.38	367.96	98.64	**0.51**	1.57
	0.3	3,094,828	9,197.77	5,201.25	1,806.24	**5.42**	1.75
	0.5	917,376,496	> 24 h	> 24 h	> 24 h	**2,144.31**	2.34
	0.1	99,062	179.80	143.03	19.39	**0.21**	2.12
1,000	0.2	1,183,584	3,750.59	4,486.30	829.52	**2.25**	1.90
	0.3	15,362,096	> 24 h	> 24 h	20,615.89	**33.18**	2.16
2,000	0.1	747,300	6,384.56	10,149.20	665.59	**2.32**	3.10
3,000	0.1	2,945,211		> 24 h	5,905.18	**11.26**	3.82
5,000	0.1	18,483,855		> 24 h	> 24 h	**86.60**	4.67
	0.001	49,738		282.40	13.30	**10.86**	218.34
	0.003	141,651		2,335.88	111.78	**11.18**	78.93
	0.005	215,129		6,138.69	364.29	**11.74**	54.57
10,000	0.01	348,552		22,426.63	645.75	**14.78**	42.40
	0.03	3,738,814		> 24 h	11,315.03	**41.29**	11.04
	0.05	12,139,182		> 24 h	> 24 h	**109.78**	9.04
	0.07	42,572,404		> 24 h	> 24 h	**338.23**	7.94
	0.1	229,786,397		> 24 h	> 24 h	**1,825.45**	7.94

local regions. In other words, given n and r, a locally random graph with n vertices is generated such that different vertices v_i and v_j are adjacent with a probability 1/2 if $i + n - j \pmod n \leqslant r$ or $j + n - i \pmod n \leqslant r$.

Table 3 shows the results for (a) $r = 10$ and (b) $r = 30$.

Table 4 shows the CPU time for sparse locally random graphs with 10,000 vertices for several values of r.

From the above experimental results, it is observed that CLIQUES is considerably faster than CLIQUE. Further, it is also confirmed to be considerably faster than the algorithm of Tsukiyama et al. [26] via the computational results in Section 8 of [14]. CLIQUES is faster than AMC and AMC* for most of the graphs tested here, while AMC and AMC* can be faster than CLIQUES when the number of maximal cliques is extremely small (e.g., c-fat200-5, c-fat500-10). AMC* can be faster than CLIQUES when the input graph is very sparse with localized edges, especially if Δ^* is chosen appropriately according to the individual graph. Makino and Uno [14] reported interesting results for very large sparse

Table 2
CPU time (sec) for Moon–Moser (M–M) graphs and DIMACS benchmark graphs

Graphs		CLIQUE	AMC	AMC*	CLIQUES	/cliques
Name (n, density)	♯cliques	[7]	[14]	[14]		
M-M-30 (30, 0.931)	59,049	1.89	0.12	0.15	**0.020**	0.34
M-M-45 (45, 0.954)	14,348,907	1,309.23	46.44	67.47	**4.80**	0.33
M-M-48 (48, 0.957)	43,046,721	5,057.65	153.21	224.41	**12.41**	0.29
M-M-51 (51, 0.960)	129,140,163	16,532.50	496.76	740.57	**32.95**	0.26
M-M-60 (60, 0.966)	3,486,784,401	$>$ 24 h	16,585.75	26,152.31	**894.90**	0.26
M-M-63 (63, 0.968)	10,460,353,203	$>$ 24 h	47,817.37	$>$ 24 h	**2,666.90**	0.25
MANN_a9 (45, 0.927)	590,887	52.64	2.24	2.93	**0.23**	0.39
brock200_2 (200, 0.496)	431,586	181.42	75.16	35.91	**0.65**	1.51
c-fat200-5 (200, 0.426)	7	0.30	**0.0029**	0.0031	0.0054	771.4
c-fat500-10 (500, 0.374)	8	6.62	**0.025**	0.028	0.058	7250.0
hamming6-2 (64, 0.905)	1,281,402	301.00	11.97	16.98	**1.21**	0.94
hamming6-4 (64, 0.349)	464	0.018	0.0086	0.0056	**0.00043**	0.93
johnson8-4-4 (70, 0.768)	114,690	13.85	1.82	1.95	**0.11**	0.96
johnson16-2-4 (120, 0.765)	2,027,025	907.51	150.68	153.42	**4.38**	2.16
keller4 (171, 0.649)	10,284,321	3,446.61	1,145.66	490.76	**4.98**	0.49
p_hat300-1 (300, 0.244)	58,176	19.23	13.52	3.52	**0.07**	1.20
p_hat300-2 (300, 0.489)	79,917,408	$>$ 24 h	16,035.71	4,130.20	**99.72**	1.25

bipartite graphs obtained from real world data. Note in CLIQUES, that the ratio of CPU time to the number of cliques varies depending on the graphs; the variation is generally not large, except for a few cases.

6. Concluding remarks

We have presented an algorithm for generating all the 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 expected to be very complicated to analyze the time complexity of our algorithm theoretically as a function of the number of maximal cliques in the graph.

The algorithm CLIQUES is very simple and consequently easy to implement. Our computational experiments demonstrate that CLIQUES runs very fast in practice.

In addition, our depth-first search algorithm can be coupled with some heuristics to yield fast algorithms MCQ and others for finding *a maximum* clique [22]. They have been successfully applied to interesting problems in bioinformatics [1–4], the design of DNA and RNA sequences for biomolecular computation [11], and other problems.

Table 3
CPU time (sec) for sparse locally random graphs

Graphs			CLIQUE	AMC	AMC*	CLIQUES	/cliques
n	Density	♯cliques	[7]	[14]	[14]		
(a) $r = 10$							
1,000	0.010	4,487	2.38	0.45	0.04	**0.016**	3.57
2,000	0.0051	9,369	18.76	4.70	0.38	**0.075**	8.01
4,000	0.0025	19,166		31.48	1.55	**0.86**	44.87
6,000	0.0017	29,192		88.91	3.56	**3.30**	113.04
8,000	0.0013	39,179		160.16	**6.25**	**6.25**	159.52
10,000	0.0010	48,975		261.27	**9.51**	10.49	214.19
(b) $r = 30$							
1,000	0.030	13,043	8.93	3.38	0.38	**0.025**	1.92
2,000	0.016	25,803	57.13	44.79	1.49	**0.10**	3.88
4,000	0.0075	53,059		243.66	9.31	**0.89**	16.77
6,000	0.0050	81,145		693.60	25.35	**3.11**	38.33
8,000	0.0038	110,354		1,271.13	46.92	**5.98**	54.19
10,000	0.0030	139,304		2,075.70	73.55	**10.02**	71.93

Table 4
CPU time (sec) for sparse locally random graphs with $n = 10,000$

Graphs			AMC	AMC*	CLIQUES	/cliques
r	Density	♯cliques	[14]	[14]		
10	0.001	48,975	261.27	**9.51**	10.49	214.19
20	0.002	95,120	952.25	49.45	**10.20**	107.23
40	0.004	181,424	3,601.09	130.76	**9.90**	54.57
80	0.008	386,360	14,448.21	431.20	**10.95**	28.34
120	0.012	746,852	35,866.69	530.53	**12.97**	17.37
160	0.016	1,408,360	> 24 h	1,066.62	**16.85**	11.96
240	0.024	4,306,123	> 24 h	4,350.94	**33.75**	7.84
320	0.032	11,488,405	> 24 h	15,655.05	**65.06**	5.66
480	0.048	66,513,724	> 24 h	> 24 h	**293.97**	4.42

Acknowledgments

The authors are grateful to T. Nakagawa for his contributions to the computational experiments [18]. They would also like to acknowledge useful comments by S. Tsukiyama, T. Nishizeki, T. Uno, K. Makino, E. Harley, D. Hochbaum, E.L. Lawler, R. Karp, J. Tarui, T. Akutsu, and the referees. 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 the Research Fund of the University of Electro-Communications. The research was also provided with a grant by Funai Foundation for Information Technology.

Appendix

A.1. Proof of Theorem 1 (Correctness of the algorithm CLIQUES)

As in Fig. 1, let $Q = \{p_1, p_2, \ldots, p_d\}$ be a complete subgraph that has been found at some stage, and consider a set of vertices

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

where $SUBG = V$ and $Q = \emptyset$ when $d = 0$.

Assume that

$$SUBG = FINI \cup CAND \quad (\neq \emptyset),$$

and that all and only maximal cliques containing $Q \cup \{q'\}$ have been generated without duplication for every $q' \in FINI$. Then we consider EXPAND(*SUBG*, *CAND*), and let a vertex u in *SUBG* be chosen at statement 4 in Fig. 2. In what follows, *CAND* stands exclusively for this *initial* value at the first entrance to statement 5. Further, let

$$CAND - \Gamma(u) = \{v_1, v_2, \ldots, v_k\} \neq \emptyset$$

and vertex q at statement 6 be chosen in the order $v_1, v_2, \ldots, v_k$. Let

$$\begin{aligned} SUBG_{v_i} &= SUBG \cap \Gamma(v_i), \\ CAND_{v_i} &= CAND \cap \Gamma(v_i), \quad \textit{and} \\ CAND^{(i)} &= CAND - \{v_1, v_2, \ldots, v_{i-1}\}. \end{aligned}$$

(See Fig. 5.)

Then, we have the next claim:

Claim A.1. (i) *For every $v_i \in CAND - \Gamma(u)$ $(1 \leqslant i \leqslant k)$ chosen as q at statement* 6, *the following holds*:

The expansion from vertex v_i by an application of EXPAND(SUBG, $CAND^{(i)}$) generates all and only maximal cliques without duplication that contain $Q \cup \{v_i\}$ and does not contain any vertex in $FINI \cup \{v_1, v_2, \ldots, v_{i-1}\}$.

(ii) *When all the maximal cliques in $G(Q \cup \{q\})$, $q \in FINI \cup (CAND - \Gamma(u))$ have finished being generated*, *we have no new maximal cliques that contain Q.*

Proof of Claim A.1. We prove the claim by induction on $n = |SUBG|$.

Basis: In the case $1 \leqslant n \leqslant 2$, the properties are easily confirmed to hold by the analysis of each case.

Induction step: We assume that the properties hold for all positive integers $n \leqslant N$, and prove that they also hold for $n = N + 1$.

(i) First, we are concerned with the case where $i = 1$ and $CAND^{(1)} = CAND$.

(a) Case $SUBG_{v_1} = \emptyset$: "*clique*," is printed after "v_1,". The generated set of vertices $Q \cup \{v_1\}$ is clearly a *maximal* clique and does not contain any vertex in *FINI*.

(b) Case $SUBG_{v_1} \neq \emptyset$: let a vertex $u' \in SUBG_{v_1}$ be chosen at statement 4, and consider EXPAND($SUBG_{v_1}$, $CAND_{v_1}$).

  (b-1) If $CAND_{v_1} - \Gamma(u') \neq \emptyset$, then application of EXPAND($SUBG_{v_1}$, $CAND_{v_1}$) guarantees to generate without duplication all and only maximal cliques that contain $Q \cup \{v_1\}$ and does not contain any vertex in $FINI = SUBG - CAND$, since $|SUBG_{v_1}| \leqslant N$ and the induction hypothesis then holds true for $SUBG_{v_1}$.

  (b-2) If $CAND_{v_1} - \Gamma(u') = \emptyset$, application of EXPAND($SUBG_{v_1}$, $CAND_{v_1}$) does nothing but prints "*back*," immediately after "v_1,". Then, the set of vertices $Q \cup \{v_1\}$ is *not* regarded as a *maximal clique* since $SUBG_{v_1} \neq \emptyset$. This is because there exist no *new* maximal cliques containing $Q \cup \{v_1\}$ than the previously generated ones.

After the application of EXPAND($SUBG_{v_1}$, $CAND_{v_1}$), the vertex v_1 is moved from *CAND* to *FINI* at statement 11. If $CAND - \Gamma(u) - \{v_1\} \neq \emptyset$, then we choose v_i at the following stages step by step for $i = 2, 3, \ldots, k$, and the above argument follows.

(ii) All maximal cliques that contain Q are in $Q \cup SUBG$. We are under the assumption that all maximal cliques in $G(Q \cup \{q\})$, $q \in SUBG - \Gamma(u)$ have finished being generated, then any remaining possibility of a new maximal clique R is such that $R = Q \cup S$ for some $S \subseteq SUBG \cap \Gamma(u)$. However, $R \cup \{u\}$ is a larger clique, and this implies that R is not *maximal*. Hence, the result follows.

Summarizing the above, the claim has been shown to hold for any positive integer $n = |SUBG|$ as a consequence of induction. □

Now if we let $Q = \emptyset$ $(d = 0)$ and $SUBG = CAND = V$ $(FINI = \emptyset)$, then consideration of $CAND - \Gamma(u) = \{v_1, v_2, \ldots, v_k\}$ gives the proof of Theorem 1. □

Lemma A.2 (*For the first part of Proof of Theorem 3 in Section 4*). *Consider the following continuous function $R(x)$ with x being a real variable*:

$$R(x) = C3^{x/3} - Q(x),$$

where

$$Q(x) = q_1x^2 + q_2x + q_3, \text{ with } q_1,\ q_2, \text{ and } q_3 \text{ as in Theorem 3.}$$

If $C \geqslant C_1 = 3q_2/\ln 3$, then $R(x)$ is monotone increasing for $x \geqslant 0$.

Proof. Differentiating the function $R(x)$, we have

$$\mathrm{d}R(x)/\mathrm{d}x = \ln 3 \cdot C3^{x/3-1} - 2q_1x - q_2.$$

In addition,

$$\mathrm{d}^2R(x)/\mathrm{d}x^2 = (\ln 3)^2 \cdot C3^{x/3-2} - 2q_1 \geqslant (\ln 3)^2 \cdot C_13^{x/3-2} - 2q_1 = \{(3\ln 3/2)3^{x/3} - 1\} \cdot p_1 > 0 \quad \text{for } x \geqslant 0,$$

since $p_1 > 0$. Hence, $\mathrm{d}R(x)/\mathrm{d}x$ is monotone increasing for $x \geqslant 0$. Thus, for $x \geqslant 0$,

$$\mathrm{d}R(x)/\mathrm{d}x \geqslant [\mathrm{d}R(x)/\mathrm{d}x]_{x=0} = \ln 3 \cdot C/3 - q_2 \geqslant \ln 3 \cdot C_1/3 - q_2 = 0.$$

Therefore, $R(x)$ is monotone increasing for $x \geqslant 0$. □

Lemma A.3 (*For the denominator of inequality (6) in Proof of Theorem 3 in Section 4*). *For all positive integers $k \neq 3$, it holds that*

$$k3^{-k/3} \leqslant 2 \cdot 3^{-2/3}.$$

Proof. Consider the following continuous function $f(x)$ with x being a real variable:

$$f(x) = x3^{-x/3}.$$

Here, we have $\mathrm{d}f(x)/\mathrm{d}x = 3^{-x/3}(1 - x \cdot \ln 3/3)$. Then $\mathrm{d}f(x)/\mathrm{d}x > 0$ for $x < 3/\ln 3$, $\mathrm{d}f(x)/\mathrm{d}x < 0$ for $x > 3/\ln 3$, and $[\mathrm{d}f(x)/\mathrm{d}x]_{x=3/\ln 3} = 0$. Subsequently, $f(x)$ is monotone increasing for $x < 3/\ln 3 = 2.7307\ldots$, $f(x)$ is monotone decreasing for $x > 3/\ln 3$, and $f(x) \leqslant f(3/\ln 3)$ for all x. In addition, $f(2) = 0.9614\ldots > f(4) = 0.9244\ldots$. Thus, the result is obtained. □

References

[1] T. Akutsu, M. Hayashida, E. Tomita, J. Suzuki, K. Horimoto, Protein threading with profiles and constraints, Proc. IEEE Symp. Bioinform. Bioeng. (2004) 537–544;
T. Akutsu, M. Hayashida, D.K.C. Bahadur, E. Tomita, J. Suzuki, K. Horimoto, Dynamic programming and clique based approaches for protein threading with profiles and constraints, IEICE Trans. Fund. Electron. Commun. Comput. Sci. E89-A (2006) 1215–1222.

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

[3] D.K.C. Bahadur, E. Tomita, J. Suzuki, T. Akutsu, Protein side-chain packing problem: a maximum edge-weight clique algorithmic approach, J. Bioinform. Comput. Biol. 3 (2005) 103–126.

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

[5] I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo, The maximum clique problem, in: D.-Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Supplement Vol. A, Kluwer Academic Publishers, Dordrecht, 1999, pp. 1–74.

[6] C. Bron, J. Kerbosch, Algorithm 457, finding all cliques of an undirected graph, Comm. ACM 16 (1973) 575–577.

[7] N. Chiba, T. Nishizeki, Arboricity and subgraph listing algorithms, SIAM J. Comput. 14 (1985) 210–223.

[8] M. Hattori, Y. Okuno, S. Goto, M. Kanehisa, Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways, J. Amer. Chem. Soc. 125 (2003) 11853–11865.

[9] H.C. Johnston, Cliques of a graph—variations on the Bron–Kerbosch algorithm, Int. J. Comput. Inform. Sci. 5 (1976) 209–238.

[10] D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, American Mathematical Society, Providence, RI, 1996.

[11] S. Kobayashi, T. Kondo, K. Okuda, E. Tomita, Extracting globally structure free sequences by local structure freeness, Proc. DNA 9 (2003) 206.

[12] I. Koch, Enumerating all connected maximal common subgraphs in two graphs, Theoret. Comput. Sci. 250 (2001) 1–30.

[13] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, SIAM J. Comput. 9 (1980) 558–565.

[14] K. Makino, T. Uno, New algorithms for enumerating all maximal cliques, SWAT 2004, Lecture Notes in Computer Science, Vol. 3111, 2004, pp. 260–272.

[15] S. Mohseni-Zadeh, P. Brézellec, J.-L. Risler, Cluster-C, an algorithm for the large-scale clustering of protein sequences based on the extraction of maximal cliques, Comput. Biol. Chem. 28 (2004) 211–218.

[16] S. Mohseni-Zadeh, A. Louis, P. Brézellec, J.-L. Risler, PHYTOPROT: a database of clusters of plant proteins, Nucleic Acids Res. 32 (2004) D351–D353.

[17] J.W. Moon, L. Moser, On cliques in graphs, Israel J. Math. 3 (1965) 23–28.

[18] T. Nakagawa, E. Tomita, Experimental comparisons of algorithms for generating all maximal cliques, Technical Report of IPSJ, 2004-MPS-52, 2004, pp. 41–44.

[19] P.M. Pardalos, J. Xue, The maximum clique problem, J. Global Optim. 4 (1994) 301–328.

[20] J.M. Robson, Algorithms for maximum independent sets, J. Algorithms 7 (1986) 425–440.

[21] R.E. Tarjan, A.E. Trojanowski, Finding a maximum independent set, SIAM J. Comput. 6 (1977) 537–546.

[22] E. Tomita, T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique, DMTCS 2003, Lecture Notes in Computer Science, Vol. 2731, 2003, pp. 278–289;
E. Tomita, T. Kameda, An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, J. Global Optim. (2006), in press;
Y. Sutani, E. Tomita, Computational experiments and analyses of a more efficient algorithm for finding a maximum clique, Technical Report of IPSJ, 2005-MPS-57, 2005, pp. 45–48, and others.

[23] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for finding all the cliques, Technical Report of the University of Electro-Communications, UEC-TR-C5(2), 1988.

[24] E. Tomita, A. Tanaka, H. Takahashi, An optimal algorithm for finding all the cliques, Technical Report of IPSJ, 1989-AL-12, 1989, pp. 91–98.

[25] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques, COCOON 2004, Lecture Notes in Computer Science, Vol. 3106, 2004, pp. 161–170.

[26] S. Tsukiyama, M. Ide, H. Ariyoshi, I. Shirakawa, A new algorithm for generating all the maximal independent sets, SIAM J. Comput. 6 (1977) 505–517.