Chapter from the book *Biomedical Engineering, Trends in Electronics, Communications and Software*
Downloaded from: http://www.intechopen.com/books/biomedical-engineering-trends-in-electronics-communications-and-software

Efficient Algorithms for Finding Maximum and Maximal Cliques: Effective Tools for Bioinformatics

Etsuji Tomita[1], Tatsuya Akutsu[2] and Tsutomu Matsunaga[3]
[1]*The University of Electro-Communications &*
The Research and Development Initiative, Chuo University, Tokyo
[2]*Kyoto University, Kyoto*
[3]*NTT DATA Corporation, Tokyo*
Japan

1. Introduction

Many problems can be formulated as graphs where a graph consists of a set of vertices and a set of edges, in which the vertices stand for objects in question and the edges stand for some relations among the objects. A clique is a subgraph in which all pairs of vertices are mutually adjacent. Thus, a maximum clique stands for a maximum collection of objects which are mutually related in some specified criterion. The so called maximum clique problem is one of the original 21 problems shown to be NP-complete by R. Karp (19). Therefore, it is strongly believed that the maximum clique problem is not solvable easily, *i.e.*, it is not solvable in polynomial-time. Nevertheless, much work has been done on this problem, experimentally and theoretically. It attracts much attention especially recently since it has found many practical applications to bioinformatics (see, e.g., (2; 15; 27; 28; 37; 3; 9; 4; 8; 14; 55; 23; 25; 22; 13)) and many others (see, e.g., excellent surveys (34; 5), and (17; 20; 31; 49; 54; 51)).
This chapter presents efficient algorithms for finding a maximum clique and maximal cliques as effective tools for bioinformatics, and shows our successful applications of these algorithms to bioinformatics.

2. Preliminaries

(1) We are concerned with a simple undirected graph $G = (V, E)$ with a finite set V of vertices and a finite set E of *unordered* pairs $(v,w)(= (w,v))$ of distinct vertices called edges. V is considered to be *ordered*, and the i-th element in V is denoted by $V[i]$. A pair of vertices v and w are said to be adjacent if $(v,w) \in E$.
(2) For a vertex $v \in V$, let $\Gamma(v)$ be the set of all vertices that are adjacent to v in $G = (V, E)$, *i.e.*, $\Gamma(v) = \{w \in V | (v,w) \in E\}$. We call $|\Gamma(v)|$, *i.e.*, the number of vertices adjacent to a vertex v, the degree of v. In general, for a set S, the number of elements in 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 state that Q is a clique. In particular, a clique which is not properly contained in any other clique is called *maximal*. A maximal clique with the maximum size is called a

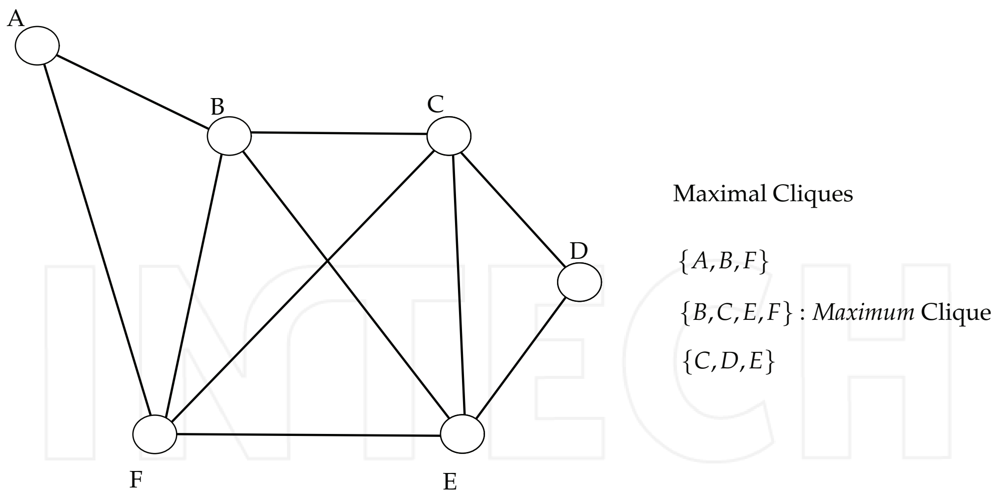


Fig. 1. Example graph

maximum clique. The number of vertices of a maximum clique in an induced subgraph $G(R)$ is denoted by $\omega(R)$.
Consider an example graph $G_0 = (V_0, E_0)$ in Fig. 1, where $V_0 = \{A, B, C, D, E, F\}$ and $E_0 = \{(A, B), (A, F), (B, C), (B, E), (B, F), (C, D), (C, E), (C, F), (D, E), (E, F)\}$. All maximal cliques are $\{A, B, F\}, \{B, C, E, F\}$ and $\{C, D, E\}$, where $\{B, C, E, F\}$ is a maximum clique of size 4. Note that $\{B, C, E\}$ is a clique, but is not a *maximal* clique since it is contained in a larger clique $\{B, C, E, F\}$.

3. Efficient algorithms for finding a maximum clique

3.1 A basic algorithm

One standard approach for finding a maximum clique is based on the branch-and-bound depth-first search method.
Our algorithm 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. More precisely, we maintain global variables Q, Q_{max}, where Q consists of vertices of a current clique, Q_{max} consists of vertices of the largest clique found so far. Let $R \subseteq V$ consist of candidate vertices which 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 and add p to Q ($Q := Q \cup \{p\}$). Then we compute $R_p := R \cap \Gamma(p)$ as the new set of candidate vertices. This procedure (EXPAND()) is applied recursively while $R_p \neq \emptyset$. Note here that **if** $|Q| + |R| \leq |Q_{max}|$ **then** $Q \cup R$ can contain only a clique that is smaller than or equal to $|Q_{max}|$, hence searching for R can be pruned in this case. This is a *basic bounding condition*.
When $R_p = \emptyset$ is reached, Q constitutes a *maximal* clique. If Q is maximal and $|Q| > |Q_{max}|$ holds, Q_{max} is replaced by Q. We then backtrack by removing p from Q and R. We select a new vertex p from the resulting R and continue the same procedure until $R = \emptyset$.
This is a well known basic algorithm for finding a maximum clique (see, e.g., (12; 10)) and is shown in detail in Fig. 2 as Algorithm BasicMC. The process can be represented by a *search*

```
procedure BasicMC ($G = (V, E)$)
begin
    global $Q := \emptyset$;
    global $Q_{max} := \emptyset$;
    EXPAND($V$);
    output $Q_{max}$
end {of BasicMC}

procedure EXPAND($R$)
begin
    while $R \neq \emptyset$ do
        $p :=$ a vertex in $R$;
        if $|Q| + |R| > |Q_{max}|$ then
            $Q := Q \cup \{p\}$;
            $R_p := R \cap \Gamma(p)$;
            if $R_p \neq \emptyset$ then EXPAND($R_p$)
            else (i.e., $R_p = \emptyset$) if $|Q| > |Q_{max}|$ then $Q_{max} := Q$ fi
            fi
            $Q := Q - \{p\}$
            else return
        fi
        $R := R - \{p\}$
    od
end {of EXPAND}
```

Fig. 2. Algorithm BasicMC

forest that is similar to Fig. 3 (b).

3.2 Algorithm MCQ

We present a very simple and efficient algorithm MCQ (46) that is a direct extension of the previous BasicMC.

3.2.1 Pruning

One of the most important points to improve the efficiency of BasicMC is to strengthen the *bounding condition* in order to prune unnecessary searching.

For a set R of vertices, let $\chi(R)$ be the chromatic number of R, *i.e.*, the *minimum* number of colors so that all pairs of adjacent vertices are colored by different colors, and $\chi'(R)$ be an *approximate* chromatic number of R, *i.e.*, a number of colors so that all pairs of adjacent vertices are colored by different colors.

Then we have that

$$\omega(R) \leq \chi(R) \leq \chi'(R) \leq |R|.$$

While obtaining $\chi(R)$ is also NP-hard, an appropriate $\chi'(R)$ could be a better upper bound of $\omega(R)$ than $|R|$, and might be obtained with low overhead. Here, we employ very simple *greedy* or *sequential* approximate coloring to the vertices of R, as introduced in (42; 12; 43). Let positive integral numbers 1, 2, 3, ... stand for colors red, green, yellow, ... *Coloring* is

also called *Numbering*. For each $p \in R$, *sequentially* from the first vertex to the last, we assign a positive integral *Number* $No[p]$ which is as small as possible. That is, when the vertices in $R = \{p_1, p_2, \ldots, p_m\}$ are arranged in this order, first let $No[p_1] = 1$, and subsequently, let $No[p_2] = 2$ if $p_2 \in \Gamma(p_1)$ else $No[p_1] = 1, \ldots$, and so on. After *Numbers* are assigned to all vertices in R, we sort these vertices in *ascending order* with respect to their *Numbers*.
We select p as the last (rightmost) vertex in R (at the 4th line in **procedure** EXPAND(R) in Fig. 2), then $No[p] = \mathrm{Max}\{No[q] \mid q \in R\}$, that is an approximate chromatic number of R. So, we replace the basic bounding condition:

$$\textbf{if } |Q| + |R| > |Q_{max}| \textbf{ then}$$

in Fig. 2 BasicMC, by the following *new bounding condition*:

$$\textbf{if } |Q| + No[p] > |Q_{max}| \textbf{ then}.$$

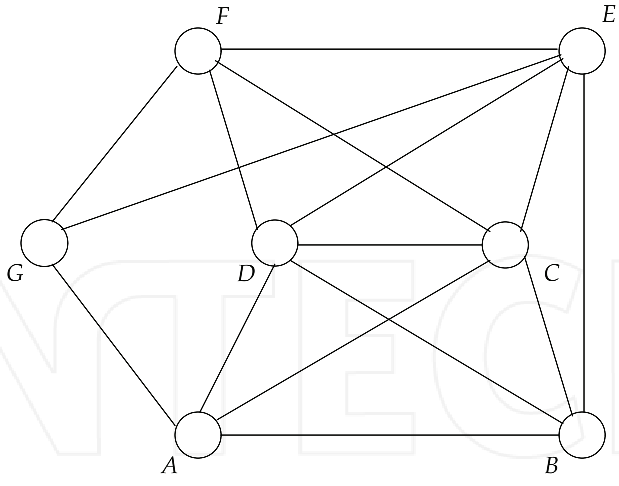
Numbering is applied every time prior to application of EXPAND().
Since the greedy Numbering to R is carried out in $O(|R|^2)$-time and is not so time-consuming, the new bounding condition can be very effective to reduce the search space with low overhead.

3.2.2 Initial sorting and simple numbering

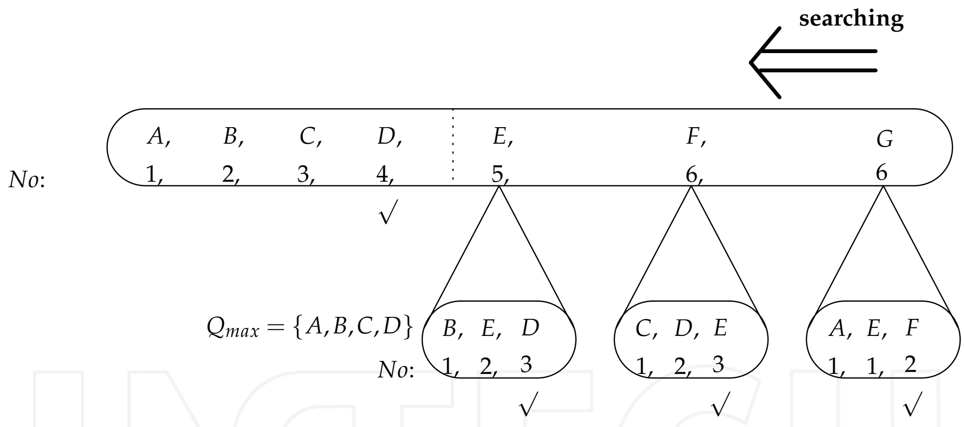
We have shown in (12) that both search space and overall running time are reduced when one sorts the vertices in an ascending order with respect to their degrees prior to the application of a branch-and-bound depth-first search algorithm for finding a maximum clique. Carraghan and Pardalos (10) also employ a similar technique successfully. Therefore, at the beginning of our algorithm, we sort vertices in V in a descending order with respect to their degrees. This means that a vertex with the minimum degree is selected at the beginning of the **while** loop in EXPAND() in Fig. 2 since the selection of p is from the last (rightmost) to the first (leftmost).
Furthermore, we initially assign *Numbers* to the vertices in V simply, so that $No[V[i]] = i$ for $i \leq \Delta(G)$, where $\Delta(G)$ is the maximum degree of G, and $No[V[i]] = \Delta(G) + 1$ for $\Delta(G) + 1 \leq i \leq |V|$. This initial *Number* has the desired property that $No[p] \geq \omega(V)$ for any p in V **while** $V \neq \emptyset$. Thus, this simple initial *Number* suffices.
This completes our explanation of the algorithm MCQ. See (46) for further details.

3.3 Algorithm MCR

Algorithm MCR (48) is an improved version of MCQ, where improvements are mainly for the *initial* sorting of the vertices. First, we alter the order of the vertices in $V = \{V[1], V[2], \ldots, V[n]\}$ so that in a subgraph of $G = (V, E)$ induced by a set of vertices $V' = \{V[1], V[2], \ldots, V[i]\}$, it holds that $V[i]$ always has the minimum degree in $\{V[1], V[2], \ldots, V[i]\}$ for $1 \leq i \leq |V|$. While the resulting order is identical to that of (10), it should be noted that time-consuming computation of the degree of vertices is carried out *only at the beginning* of MCR as in MCQ, and hence the overhead of the overall selection of vertices is *very small*, too. Here, the degrees of adjacent vertices are also taken into consideration.
Numbering of the vertices is carried out in a similar way to MCQ, but more closely. The above considerations lead to an improved clique finding algorithm MCR.
Example run We show an example run of MCR to an input graph G_1 given in Fig. 3 (a).
In the initial sorting, vertex G with the minimum degree is selected at first. Then, vertices F and E follow after G. Now the remaining vertices A, B, C, D are of the same minimum degree,



(a) An input graph G_1



(b) A search forest of G_1

Fig. 3. Example run of MCR

and then *Numbering* is applied to this set of vertices to have $No[A] = 1$, $No[B] = 2$, $No[C] = 3$, $No[D] = 4$. The other vertices are numbered as $No[E] = 4 + 1 = 5$, $No[F] = 6$ $(= \Delta(G_1) + 1)$, $No[G] = 6$. In addition, we have that $Q_{max} = \{A, B, C, D\}$ (of size 4), since every degree of the vertices in the induced subgraph $G_1(\{A, B, C, D\})$ is $3 = 4 - 1$.
The result of the initial sorting of vertices is shown at the top of Fig. 3 (b) and the corresponding numbers are just below these vertices.
Subsequently, in EXPAND( ), the rightmost vertex G is selected to have $Q = \{G\}$, $R_G = \Gamma(G) = \{A, E, F\}$. These vertices A, E, F are numbered 1,1,2 as shown in the second row of Fig. 3 (b).

Here, $|Q| + No[F] = |\{G\}| + 2 = 3 < |Q_{max}| = 4$, then prune ($\surd$: checkmark in Fig. 3 (b)). The searching proceeds from the right to the left as shown in Fig. 3 (b). As a result, the maximum clique in G_1 is $Q_{max} = \{A, B, C, D\}$.

3.4 Algorithm MCS

Algorithm MCS (39; 49; 50) is a further improved version of MCR.

3.4.1 New approximate coloring

When vertex r is selected, if $No[r] \leq |Q_{max}| - |Q|$ then it is not necessary to search from vertex r by the *bounding condition*, as mentioned in Sect. 3.2.1. 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 $No[p] > |Q_{max}| - |Q|$ ($\overset{def}{=} No_{th}$) (No_{th} stands for $No_{threshold}$), we attempt to change its *Number* in the following manner (16). Let No_p denote the original value of $No[p]$.
[Re-NUMBER p]
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 *Numbers* 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*.

3.4.2 Adjunct ordered set of vertices for approximate coloring

The ordering of vertices plays an important role in the algorithm as demonstrated in (12; 10; 46; 48). In particular, the procedure *Numbering* strongly depends on the order of vertices, since it is a *sequential* coloring. In our new algorithm, we sort the vertices in the same way as in MCR (48) at the first stage. However, the vertices are *disordered* in the succeeding stages owing to the application of Re-NUMBER. In order to avoid this difficulty, we employ another *adjunct ordered set* V_a *of vertices for approximate coloring* that preserves the order of vertices appropriately sorted in the first stage. Such a technique was first introduced in (38).
We apply *Numbering* to vertices from the first (leftmost) to the last (rightmost) in the order maintained in V_a, while we select a vertex in the ordered set R for *searching*, beginning from the last (rightmost) vertex and continuing up to the first (leftmost) vertex.
An improved MCR obtained by introducing only the technique (38) in this section is named MCR*.

3.4.3 Reconstruction of the adjacency matrix

Each graph is stored as an adjacency matrix in the computer memory. Sequential *Numbering* is carried out according to the initial order of vertices in the adjunct ordered set V_a, as described in Sect. 3.4.2. 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 (41) results in a more effective use of the cache memory.
The new algorithm obtained by introducing all the techniques described in Sects. 3.4.1–3.4.3 in MCR is named MCS. Table 1 shows the running time required to solve some DIMACS

Graph	dfmax (18)	New (33)	ILOG (35)	MCQ (46)	MCR (48)	MCS (50)
brock400_1	22,051		8,401	1,783	1,771	693
block800_1	$> 10^5$		$> 10,667$	18,002	17,789	9,347
MANN_a27	$> 10^5$	$> 2,232$	14	5.4	2.5	0.8
MANN_a45	$> 10^5$		$> 10,667$	4,646	3,090	281
p_hat500-2	133	96	24	4.0	3.1	0.7
p_hat1000-2	$> 10^5$		12,478	2,844	2,434	221
san200_0.9_3	42,648		135	10	0.16	0.06
san400_0.7_2	$> 10^5$	113	50	1.0	0.3	0.1
san400_0.9_1	$> 10^5$		1,259	46.3	3.4	0.1
gen200_p0.9_44	48,262				5.39	0.47
gen400_p0.9_55					5,846,951	58,431
gen400_p0.9_65					$> 2.5 \times 10^7$	151,597
C250.9	$> 10^5$				44,214	3,257

Table 1. Comparison of the running time [sec]

benchmark graphs (18) by representative algorithms dfmax (18), New (33), ILOG (35), MCQ, MCR, and MCS, taken from (50). (10^5 seconds $\simeq$ 1.16 days).
Our user time (T_1) *in (50) for DIMACS benchmark instances:* `r100.5`, `r200.5`, `r300.5`, `r400.5`, and `r500.5` are 1.57×10^{-3}, 4.15×10^{-2}, 0.359, 2.21, and 8.47 seconds, respectively. (*Correction*: These values described in the Appendix of (50) should be corrected as shown above. However, other values in (50) are computed based on the above correct values, hence other changes in (50) are *not* necessary.)
While MCR* obtained by introducing the adjunct set V_a of vertices for approximate coloring in Sect. 3.4.2 is almost always more efficient than MCR (38), combination of all the techniques in Sects. 3.4.1–3.4.3 makes it much more efficient to have MCS.
The aim of the present study is to develop a faster algorithm whose use is not confined to any particular type of graphs. 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, and hence reduce the overall running time for *dense* graphs (36; 21), but with the increase of the overall running time for *nondense* graphs. Appropriately controlled application of repeated sorting of vertices can make the algorithm more efficient for wider classes of graphs (21).
Parallel processing for maximum-clique-finding is very promising in practice (41; 53).
For practical applications, *weighted* graphs becomes more important. Algorithms for finding *maximum-weighted* cliques have also been developed. For example, see (45; 32; 30) for *vertex*-weighted graphs and (40) for *edge*-weighed graphs.

4. Efficient algorithm for generating all maximal cliques

In addition to finding only one maximum clique, generating all maximal cliques is also important and has many diverse applications.
In this section, we present a depth-first search algorithm CLIQUES (44; 47) for generating all maximal cliques of an undirected graph $G = (V, E)$, in which pruning methods are employed as in Bron and Kerbosch's algorithm (7). All maximal cliques generated are output in a tree-like form.

4.1 Algorithm CLIQUES

The basic framework of CLIQUES is almost the same as BasicMC *without the basic bounding condition*.

Here, we describe two methods to prune unnecessary parts of the search forest, which happened to be the same as in the Bron-Kerbosch algorithm (7). We regard the set $SUBG$ ($= V$ at the beginning) as 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 a subset of vertices of $SUBG$ that have been already processed by the algorithm. ($FINI$ is short for *"finished"*.) Then we denote by $CAND$ the set of remaining candidates for expansion: $CAND = SUBG - FINI$. So, we have

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

where $FINI = \emptyset$ at the beginning. Consider the subgraph $G(SUBG_q)$ with $SUBG_q = SUBG \cap \Gamma(q)$, and let

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

where $FINI_q = FINI \cap \Gamma(q)$ and $CAND_q = CAND \cap \Gamma(q)$. Then only the vertices in $CAND_q$ can be candidates for expanding the complete subgraph $Q \cup \{q\}$ to find *new* larger cliques.

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 $Q \cup \{u\}$, must contain at least one vertex $q \in SUBG - \Gamma(u)$.

Taking the previously described pruning method also into consideration, the only search subtrees to be expanded are from vertices in $(SUBG - SUBG \cap \Gamma(u)) - FINI = CAND - \Gamma(u)$. Here, in order to minimize $\mid CAND - \Gamma(u) \mid$, we choose such vertex $u \in SUBG$ to be the one which *maximizes* $\mid CAND \cap \Gamma(u) \mid$. This is *essential* to establish the *optimality* of the worst-case time-complexity of CLIQUES.

Our algorithm CLIQUES (47) for generating all maximal cliques is shown in Fig. 4. Here, 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. Instead, in addition to printing "clique" at statement 3, 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, and 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.

The output in a tree-like format is also important *practically*, since it saves space in the output file.

4.2 Time-complexity of CLIQUES

We have proved that the worst-case time-complexity is $O(3^{n/3})$ for an n-vertex graph (47). This is *optimal* as a function of n, since there exist up to $3^{n/3}$ cliques in an n-vertex graph (29). The algorithm is also demonstrated to run fast in practice by computational experiments. Table 2 shows the running time required to solve some DIMACS benchmark graphs by representative algorithms CLIQUE (11), AMC (24), AMC* (24), and CLIQUES, taken from (47).

For practical applications, enumeration of *pseudo* cliques sometimes becomes more important (52).

```
procedure CLIQUES($G$)
begin
 1 : EXPAND($V$,$V$)
end of CLIQUES

      procedure EXPAND($SUBG$, $CAND$)
      begin
 2 :    if $SUBG = \emptyset$
 3 :      then print (“clique,”)
 4 :      else $u :=$ a vertex $u$ in $SUBG$ which maximizes $| CAND \cap \Gamma(u) |$;
 5 :           while $CAND - \Gamma(u) \neq \emptyset$
 6 :               do $q :=$ a vertex in $(CAND - \Gamma(u))$;
 7 :                  print ($q$, “,”);
 8 :                  $SUBG_q := SUBG \cap \Gamma(q)$;
 9 :                  $CAND_q := CAND \cap \Gamma(q)$;
10 :                  EXPAND($SUBG_q$,$CAND_q$);
11 :                  $CAND := CAND - \{q\}$;
12 :                  print (“back,”)
               od
        fi
      end of EXPAND
```

Fig. 4. Algorithm CLIQUES

Graph	CLIQUE (11)	AMC (24)	AMC* (24)	CLIQUES (47)
brock200_2	181.4	75.2	35.9	0.7
johnson16-2-4	908	151	153	4
keller4	3,447	1,146	491	5
p_hat300-2	$> 86,400$	16,036	4,130	100

Table 2. Comparison of the running time [sec]

5. Applications to bioinformatics

5.1 Analysis of protein structures

In this subsection, we show applications of maximum clique algorithms to the following three problems on protein structure analysis: (i) *protein structure alignment*, (ii) *protein side-chain packing*, (iii) *protein threading*. Since there are many references on these problems, we only cite references that present the methods shown here. Most of other relevant references can be reached from those references. Furthermore, we present here only the definitions of the problems and reductions to clique problems. Readers interested in details such as results of computational experiments are referred to the original papers (1; 2; 3; 4; 8).

5.1.1 Protein structure alignment

Comparison of protein structures is very important for understanding the functions of proteins because proteins with similar structures often have common functions. Pairwise comparison of proteins is usually done via *protein structure alignment* using some scoring scheme, where an alignment is a mapping of amino acids between two proteins. Because of

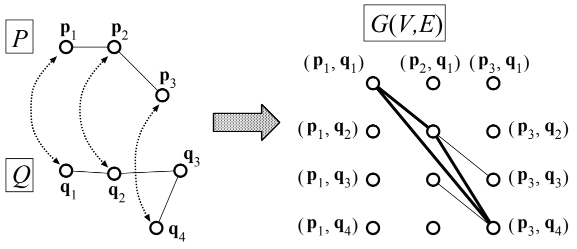

P
Q
$\mathbf{p}_1$
$\mathbf{p}_2$
$\mathbf{p}_3$
$\mathbf{q}_1$
$\mathbf{q}_2$
$\mathbf{q}_3$
$\mathbf{q}_4$
G(V,E)
$(\mathbf{p}_1, \mathbf{q}_1)$
$(\mathbf{p}_2, \mathbf{q}_1)$
$(\mathbf{p}_3, \mathbf{q}_1)$
$(\mathbf{p}_1, \mathbf{q}_2)$
$(\mathbf{p}_3, \mathbf{q}_2)$
$(\mathbf{p}_1, \mathbf{q}_3)$
$(\mathbf{p}_3, \mathbf{q}_3)$
$(\mathbf{p}_1, \mathbf{q}_4)$
$(\mathbf{p}_3, \mathbf{q}_4)$

Fig. 5. Reduction from protein structure alignment to maximum clique. Maximum clique shown by bold lines (right) corresponds to protein structure alignment shown by dotted lines (left).

its importance, many methods have been proposed for protein structure alignment. However, most existing methods are heuristic ones in which optimality of the solution is not guaranteed. Bahadur et al. developed a clique-based method for computing structure alignment under some local similarity measure (2). Let $P = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m)$ be a sequence of three-dimensional positions of amino acids (precisely, positions of Cα atoms) in a protein. Let $Q = (\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n)$ be a sequence of positions of amino acids of another protein. For two points $\mathbf{x}$ and $\mathbf{y}$, $|\mathbf{x} - \mathbf{y}|$ denotes the Euclidean distance between x and y. Let $f(x)$ be a function from the set of non-negative reals to the set of reals no less than 1.0. We call a sequence of pairs $M = ((\mathbf{p}_{i_1}, \mathbf{q}_{i_1}), \dots, (\mathbf{p}_{i_l}, \mathbf{q}_{i_l}))$ an *alignment* under *non-uniform distortion* if the following conditions are satisfied:

- $i_k < i_h$ and $j_k < j_h$ hold for all $k < h$,
- $(\forall k)(\forall h \neq k)\left(\dfrac{1}{f(r)} < \dfrac{|\mathbf{q}_{j_h} - \mathbf{q}_{j_k}|}{|\mathbf{p}_{j_h} - \mathbf{p}_{j_k}|} < f(r)\right)$,

where $r = \min\{|\mathbf{q}_{j_h} - \mathbf{q}_{j_k}|, |\mathbf{p}_{j_h} - \mathbf{p}_{j_k}|\}$. Then, protein structure alignment is defined as the problem of finding a longest alignment (*i.e.*, l is the maximum). It is known that protein structure alignment is NP-hard under this definition.
This protein structure alignment problem can be reduced to the maximum clique problem in a simple way (see Fig. 5). we construct an undirected graph $G(V, E)$ by

$$\begin{aligned} V &= \{ (\mathbf{p}_i, \mathbf{q}_j) \mid i = 1, \dots, m, \ j = 1, \dots, n \}, \\ E &= \{ \{(\mathbf{p}_i, \mathbf{q}_j), (\mathbf{p}_k, \mathbf{q}_h)\} \mid i < k, \ j < h, \ \frac{1}{f(r)} < \frac{|\mathbf{q}_h - \mathbf{q}_j|}{|\mathbf{p}_k - \mathbf{p}_i|} < f(r) \}. \end{aligned}$$

Then, it is straight-forward to see that a maximum clique corresponds to a longest alignment.

5.1.2 Protein side-chain packing

The *protein side-chain packing* problem is, given an amino acid sequence and spatial information on the main chain, to find *side-chain conformation* with the minimum potential energy. In most cases, it is defined as a problem of seeking a set of $(\chi_1, \chi_2, \dots)$ angles whose *potential energy* becomes the minimum, where positions of atoms in the main chain are fixed. This problem is important for prediction of detailed structures of proteins because such prediction methods as protein threading cannot determine positions of atoms in the side-chains. It is known that protein side-chain packing is NP-hard and thus various heuristic methods have

been proposed. Here, we briefly review a clique-based approach to protein side-chain packing (2; 3; 8).
Let $R = \{r_1, \ldots, r_n\}$ be the set of amino acid residues in a protein. Here, we only consider χ_1 angles and then assume that positions of atoms in a side-chain are rotated around the χ_1 axis. Let $r_{i,k}$ be the ith residue whose side-chain atoms are rotated by $(2\pi k)/K$ radian, where we can modify the problem and method so that the rotation angles can take other discrete values. We say that residue $r_{i,k}$ collides with the main chain if the minimum distance between the atoms in $r_{i,k}$ and the atoms in the main chain is less than a threshold L_1Å. We say that residue $r_{i,k}$ collides with residue $r_{j,h}$ if the minimum distance between the atoms in $r_{i,k}$ and the atoms in $r_{j,h}$ is less than L_2Å. We define an undirected graph $G(V;E)$ by

$$\begin{aligned} V &= \{\, r_{i,k} \mid r_{i,k} \text{ does not collide with the main chain } \}, \\ E &= \{\, \{r_{i,k}, r_{j,h}\} \mid r_{i,k} \text{ does not collide with } r_{j,h} \,\}. \end{aligned}$$

Then, it is straight-forward to see that a clique with size n corresponds to a consistent configuration of side chains (*i.e.*, side-chain conformation with no collisions). We can extend this reduction so that potential energy can be taken into account by using the *maximum edge-weighted clique* problem.

5.1.3 Protein threading

Protein threading is one of the prediction methods for three-dimensional protein structures. The purpose of protein threading is to seek for a protein structure in a database which best matches a given protein sequence (whose structure is to be predicted) using some score function.
In order to evaluate the extent of match, it is required to compute an *optimal alignment* between an amino acid sequence $S = s_1 s_2 \ldots s_n$ and a known protein structure $P = (\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m)$, where s_i and p_j denote the ith amino acid and the jth residue position, respectively. As in protein structure alignment, a sequence of pairs $((s_{i_1}, \mathbf{p}_{j_1}), (s_{i_2}, \mathbf{p}_{j_2}), \ldots, (s_{i_l}, \mathbf{p}_{j_l}))$ is called an *alignment* (or, a *threading*) between S and P if $i_k < i_h$ and $j_k < j_h$ hold for all $k < h$. Let $g(s_{i_k}, s_{i_h}, \mathbf{p}_{j_k}, \mathbf{p}_{j_h})$ give a score (e.g., pseudo energy) between residue positions of $\mathbf{p}_{j_k}$ and $\mathbf{p}_{j_h}$ when amino acids s_{i_k} and s_{i_h} are assigned to positions of $\mathbf{p}_{j_k}$ and $\mathbf{p}_{j_h}$, respectively. Then, protein threading is defined as a problem of finding an optimal alignment that minimizes the pseudo energy:

$$\sum_{k<h} g(s_{i_k}, s_{i_h}, \mathbf{p}_{j_k}, \mathbf{p}_{j_h}),$$

where we ignore gap penalties for the simplicity.
This protein threading problem can be reduced to the *maximum edge-weighted clique* problem (1; 4), which seeks for a clique that maximizes the total weight of edges in the clique. From an instance of protein threading, we construct an undirected graph $G(V,E)$ by

$$\begin{aligned} V &= \{\, (s_i, \mathbf{p}_j) \mid i = 1, \ldots, n,\ j = 1, \ldots, m \,\}, \\ E &= \{\, \{(s_i, \mathbf{p}_j), (s_k, \mathbf{p}_h)\} \mid i < k,\ j < h \,\}, \end{aligned}$$

where the weight of an edge is given by $-g(s_i, s_k, \mathbf{p}_j, \mathbf{p}_h)$. It is straight-forward to see that a maximum edge-weight clique corresponds to an optimal alignment. Though this clique-based approach is not necessarily the best for protein threading, the results of (1; 4) suggest that it is useful for protein threading with certain constraints.

5.2 Data mining for related genes in a biomedical database

In this subsection, we present an application of enumerating cliques. Readers interested in details are referred to the original paper (25).

Progress in the life sciences cannot be made without integrating biomedical knowledge on numerous genes in order to help formulate hypotheses on the genetic mechanisms behind various biological phenomena, including diseases. There is thus a strong need for a way to automatically and comprehensively search biomedical databases for related genes, such as genes in the same families and genes encoding components of the same pathways.

We constructed a graph whose vertices (nodes) were gene or disease pages, and edges were the hyperlink connections between those pages in the Online Mendelian Inheritance in Man (OMIM) database (25; 26). This work was based on the assumption that the structures of hyperlink connections correspond to the structural features of biological systems. Clique enumeration approach has been applied to a relational graph based on the assumption that relevant relationships are reflected in completely interconnected subgraphs (cliques) or nearly completely interconnected subgraphs (pseudo-cliques). We address the extraction of related genes by searching for densely connected subgraphs in a biomedical relational graph. Sets of related genes are detected by enumerating densely-connected subgraphs modeled as cliques (47) or pseudo-cliques (52).

We obtained over 20,000 sets of related genes (called 'gene modules') by enumerating cliques computationally. Table 3 shows gene sets included in typical large gene modules. The gene module in the first row is constituted by a family of chemokine genes, and the gene module in the second comprises NF-κB family genes (including RelA and RelB) and genes that form complexes with them (IκB). The gene module in the third row is made up of 'DNA repair'-related genes. The BRCA1-associated proteins; the BLM, MSH6, MSH2, and MLH1 proteins; and subunits of the RFC complex are involved in DNA repair. The genes in the module in the fourth row are related to general transcription factor (GTF) protein complexes. The gene module in the bottom row is associated with the signal transduction pathway of the inflammatory response. TNF receptor-associated factor 2 (TRAF2) is a protein that interacts with TNF receptors and is required for signal transduction. The MAP kinase kinase kinase 14 (MAP3K14) gene in this module encodes a protein that simulates NF-κB activity by binding to the TRAF2 gene product. The gene modules thus comprise various types of related genes including gene families, complexes, and pathways.

For applying gene modules to disease mechanism analysis, we assembled gene modules associated with the metabolic syndrome as an example of a typical multifactorial disease comprising obesity, diabetes, hyperlipidemia, and hypertension. The number of gene modules associated with diabetes, hyperlipidemia, hypertension, and obesity were 110, 16, 34, and 28, respectively. There were no overlaps among the modules. Then a total of 188 modules and 124 genes contained were identified. The 10 most frequent genes in the 188 modules are listed in Table 4 along with the numbers of times they were found in the modules (*i.e.*, cliques) of various sizes. As shown in the table, INS gene and LEP gene are

Gene module	Attribute
{ PPBP, SCYB6, GRO2, GRO3, IL8, SCYB10, IFNG, GRO1, PF4, SCYB5, MIG, SCYB11 }	Family
{ NFKBIA, NFKB1, NFKB2, RELA, REL, CHUK, MAP3K7, IKBKB, NFKBIB, MAP3K14, RELB }	Family & Complex
{ RFC4, RFC1, BRCA1, MSH2, MLH1, APC, RFC2, MSH6, MRE11A, BLM }	Complex
{ POLR2A, GTF2E1, GTF2B, GTF2F1, GTF2H1, TAF1, TAF10, GTF2A2, GTF2A1 }	Complex
{ TNFRSF5, NFKB1, TNF, TNFRSF1A, TNFRSF1B, CHUK, TRAF2, MAP3K14 }	Pathway

Table 3. Typical large gene modules computationally extracted as pseudo-cliques.

			Size					
Rank	Gene	Total	2	3	4	5	6	7
1	INS	29	2	6	2	18	1	0
2	LEP	27	2	6	4	12	3	0
3	POMC	16	1	0	3	10	2	0
4	PCSK1	13	0	1	2	9	1	0
5	IRS2	12	0	0	0	11	1	0
5	IGF1	12	0	1	0	10	1	0
5	INSR	12	3	4	1	4	0	0
8	IRS1	11	0	0	1	9	1	0
9	MC4R	10	0	0	1	7	2	0
10	FGF1	9	0	0	1	4	2	2

Table 4. The 10 most frequent genes in the 188 extracted modules associated with the metabolic syndrome.

the top and the 2nd, respectively. The modules of size 6 including INS gene or LEP gene were {Obesity,LEP,MC4R,POMC,AGRP,LEPR}, {Obesity,LEP,MC4R,POMC,AGRP,PCSK1} and {Diabetes,LEP,IGF1,IRS1,INS,IRS2}. Each module contains biologically plausible genes related to obesity or diabetes. By combining the 188 modules and 124 genes using the correspondence analysis, we obtained a coherent holistic picture helpful for interpreting relations among genes (25).
The comprehensive extraction of gene modules can be a potential aid to researchers in the biomedical sciences by providing a systematic methodology for interpreting relationships among genes and biological phenomena.

6. Conclusion

We have presented efficient algorithms for finding maximum and maximal cliques, and shown our successful application to bioinformatics. It is expected that these algorithms can be convenient and effective tools for much more problems in bioinformatics.

7. Acknowledgments

We should like to express our sincere gratitude to our many colleagues and (former) students who worked with us in this research. Many helpful comments by E. Harley are appreciated. We wish to thank P.M. Pardalos and his colleagues for reviewing our earlier works including (43), (44) in their surveys (34), (5). Our earlier works received considerable attention by their reviews. Thanks are also to D.S. Johnson and M. Trick for their efforts in organizing the Second DIMACS Implementation Challenge for Cliques, Coloring, and Satisfiability (18). They made it easier for us to compare the results of different algorithms carried out on various computers. This research was partially supported by Grants-in-Aid for Scientific Research Nos. 19500010, 21300047, 22500009, 22240009, and many others from the Ministry of Education, Culture, Sports, Science and Technology, Japan, a Special Grant for the Strategic Information and Communications R&D Promotion Programme (SCOPE) Project from the Ministry of Internal Affairs and Communications, Japan, and the Research Fund of the University of Electro-Communications to the Advanced Algorithms Research Laboratory. The research was also provided a grant by the Funai Foundation for Information Technology.

8. References

[1] Akutsu, T., Hayashida, M., Bahadur D.K.C, Tomita, E., Suzuki, J., Horimoto, K.: Dynamic programming and clique based approaches for protein threading with profiles and constraints, IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences, E89-A, 1215-1222 (2006). The preliminary version was presented In: Akutsu, T., Hayashida, M., Tomita, E., Suzuki, J., Horimoto, K.: Protein threading with profiles and constraints, Proc. IEEE Symp. on Bioinformatics and Bioengineering (BIBE 2004), 537-544 (2004)

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

[3] Bahadur D.K.C, Tomita, E., Suzuki, J., Akutsu, T.: Protein side-chain packing problem: A maximum edge-weight clique algorithmic approach, J. Bioinformatics and Computational Biology, 3, pp.103-126 (2005)

[4] 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)

[5] 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, Kluwer Academic Publishers, 1–74 (1999)

[6] Bradde, S., Braunstein, A., Mahmoudi, H., Tria, F., Weigt, M., Zecchina, R.: Aligning graphs and finding substructures by a cavity approach, Europhisics Letters, 89 (2010)

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

[8] Brown, J.B., Bahadur, D.K.C., Tomita, E., Akutsu, T.: Multiple methods for protein side chain packing using maximum weight cliques, Genome Informatics, 17(1), 3–12 (2006)

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

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

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

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

[13] Fukagawa, D., Tamura, T., Takasu, A., Tomita, E., Akutsu, T.: A Clique-based method for the edit distance between unordered trees and its application to analysis of glycan structure, BMC Bioinformatics, Suppl. for APBC 2011 (to appear)

[14] Han, K., Cui, G., Chen, Y.: Identifying functional groups by finding cliques and near-cliques in protein interaction networks, Proc. 2007 Frontiers in the Convergence of Bioscience and Information Technologies, 159–164 (2007)

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

[16] Higashi, T., Tomita, E.: A more efficient algorithm for finding a maximum clique based on an improved approximate coloring, Technical Report of the University of Electro-Communications, UEC-TR-CAS5 (2006)

[17] Hotta, K., Tomita, E., Takahashi, H.: A view-invariant human face detection method

based on maximum cliques. Trans. IPSJ, 44, SIG14(TOM9), 57–70 (2003)

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

[19] Karp, R.: Reducibility among combinatorial problems, In; Miller, R.E., Thatcher, J.W. (Eds.) *Comlexity of Computer Computations*, Plenum Press, New York, 85–103 (1972)

[20] Kobayashi, S., Kondo, T., Okuda, K., Tomita, E.: Extracting globally structure free sequences by local structure freeness. In: Chen, J., Reif, J. (Eds.), Proc. Ninth International Meeting on DNA Based Computers, 206 (2003)

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

[22] Li, X., Wu, M., Kwoh, C.-K. Ng, S.-K.: Computational approaches for detecting protein complexes from protein interaction networks: a survey, BMC Genomics, 11 (2010)

[23] Liu, G., Wong, L., Chua, H.-N.: Complex discovery from weighted PPI networks, Bioinformatics, 1891–1897 (2009)

[24] Makino, K., Uno, T.: New algorithms for enumerating all maximal cliques, SWAT 2004, Lecture Notes in Computer Science, 3111, 260–272 (2004)

[25] Matsunaga, T., Yonemori, C., Tomita, E., Muramatsu, M.: Clique-based data mining for related genes in a biomedical database, BMC Bioinformatics, 10:205 (2009)

[26] Matsunaga, T., Kuwata, S., Muramatsu, M.: Computational gene knockout reveals transdisease-transgene association structure, J. Bioinformatics and Computational Biology, 8, 843-866 (2010)

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

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

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

[30] Nakamura, T., Tomita, E.: Efficient algorithms for finding a maximum clique with maximum vertex weight, Technical Report of the University of Electro-Communications, UEC-TR-CAS3 (2005)

[31] Nakui, Y., Nishino, T., Tomita, E., Nakamura, T.: On the minimization of the quantum circuit depth based on a maximum clique with maximum vertex weight. Technical Report of RIMS, 1325, Kyoto University, 45–50 (2003)

[32] Östergård, P.R.J.: A fast algorithm for the maximum-weight clique problem, Nordic J. Computing, 8, 424-436 (2001)

[33] Östergård, P.R.J.: A fast algorithm for the maximum clique problem, Discrete Applied Math., 120, 197–207 (2002)

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

[35] Régin, J.-C.: Using constraint programming to solve the maximum clique problem, Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, 2833, 634-648 (2003)

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

[37] Strickland, D.M., Barnes, E., Sokol, J.S.: Optimal protein structure alignment using maximum cliques, Operations Research, 53, 389–402 (2005)

[38] 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, 45–48 (2005)
[39] Sutani, Y., Higashi, T., Tomita, E. Takahashi, S., Nakatani, H.: A faster branch-and-bound algorithm for finding a maximum clique, Technical Report of IPSJ, 2006-AL-108, 79–86 (2006)
[40] Suzuki, J., Tomita, E., Seki, T.: An algorithm for finding a maximum clique with maximum edge-weight and computational experiments, Technical Report of IPSJ, 2002-MPS-42, 45–48 (2002)
[41] Takahashi, S., Tomita, E.: Parallel computation for finding a maximum clique on shared memory computers, Technical Report of the University of Electro-Communications, UEC-TR-CAS3 (2007)
[42] Tomita, E., Yamada, M.: An algorithm for finding a maximum complete subgraph, Conference Records of the National Convention of IECE 1978, 8 (1978)
[43] Tomita, E., Kohata, Y., Takahashi, H.: A simple algorithm for finding a maximum clique, Technical Report of the University of Electro-Communications, UEC-TR-C5(1) (1988)
[44] Tomita, E., Tanaka, A. Takahashi, H.: The worst-case time complexity for finding all maximal cliques, Technical Report of the University of Electro-communications, UEC-TR-C5(2) (1988)
[45] Tomita, E. Wakai, Y., Imamatsu, K.: An efficient algorithm for finding a maximum weight clique and its experimental evaluations, Technical Report of IPSJ, 1999-MPS-27, 33–36 (1999)
[46] Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique, Discrete Math. and Theoret. Comput. Sci., Lecture Notes in Computer Science, 2731, 278–289 (2003)
[47] 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) (Awarded "Theoretical Computer Science Top Cited Article 2005-2010" by Elsevier.)
[48] Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, J. Global Optimization, 37, 95–111 (2007), J. Global Optimization, 44, 311 (2009)
[49] Tomita, E.: The maximum clique problem and its applications - Invited Lecture, Technical Report of IPSJ, 2007-MPS-67/2007-BIO-11, 21–24 (2007)
[50] Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique, WALCOM: Algorithms and Complexity, Lecture Notes in Computer Science, 5942, 191-203 (2010)
[51] Tomita, E.: Plenary Lecture: The maximum clique problem, Proc. 14th WSEAS International Conf. on Computers (vol. I), 19, Corfu Island, Greece (2010)
[52] Uno, T.: An efficient algorithm for solving pseudo clique enumeration problem, Algorithmica, 56, 3–16 (2010)
[53] Wakatsuki, M., Takahashi, S., Tomita, E.: A parallelization of an algorithm for finding a maximum clique on shared memory computers, Technical Report of IPSJ, 2008-MPS-71, 17–20 (2008)
[54] Yonemori, C., Matsunaga, T., Sekine, J., Tomita, E.: A structural analysis of enterprise relationship using cliques, DBSJ Journal, 7, 55-60 (2009)
[55] Zhang, B., Park, B.-H., Karpinets, T., Samatova, N.F.: From pull-down data to protein interaction networks and complexes with biological relevance, Bioinformatics, 24, 979–986 (2008)