Efficient Algorithms for Finding Maximum and Maximal Cliques and Their Applications

著者（英）	Etsuji Tomita
journal or publication title	Lecture Notes in Computer Science
volume	10167
page range	3-15
year	2017-02-21
URL	http://id.nii.ac.jp/1438/00008519/

Efficient Algorithms for Finding Maximum and Maximal Cliques and Their Applications

Etsuji Tomita*

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

Abstract. The problem of finding a maximum clique or enumerating all maximal cliques is very important and has been explored in several excellent survey papers. Here, we focus our attention on the step-by-step examination of a series of branch-and-bound depth-first search algorithms: Basics, MCQ, MCR, MCS, and MCT. Subsequently, as with the depth-first search as above, we present our algorithm, CLIQUES, for enumerating all maximal cliques. Finally, we describe some of the applications of the algorithms and their variants in bioinformatics, data mining, and other fields.

1. Introduction

Given an undirected graph G, a clique is a subgraph in which all pairs of vertices are mutually adjacent in G. The so-called maximum clique problem is one of the original 21 problems shown to be NP-complete by Karp [18]. The problem of finding a maximum clique or enumerating all maximal cliques in G is very important and significant work has been done on it, both theoretically and experimentally [34, 17, 7, 10, 49, 54].

An excellent review on recent various algorithms for the maximum clique problems can be found in [54] by Wu and Hao. Herein, we focus our attention on a series of branch-and-bound depth-first search algorithms — Basics [13, 43], MCQ [45], MCR [47], MCS [48], and MCT [52] — for finding a maximum clique, such that their progress can be understood easily.

Subsequently, similarly to the depth-first searches above, we present our $O(3^{n/3})$-time algorithm, CLIQUES [46], for enumerating all maximal cliques, that is optimal with respect to the number of vertices n.

Finally, we outline some of the applications of the previous algorithms or their variants to fields including bioinformatics, data mining, and image processing.

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) $(= (w, v), v \neq w)$ 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 adjacent to v in $G = (V, E)$, *i.e.*, $\Gamma(v) = \{w \in V | (v, w) \in E\}$. We call $|\Gamma(v)|$ the degree of v. 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 that is not properly contained in any other clique is called *maximal*. A maximal clique with the maximum size is called a *maximum clique*. The number of vertices of a maximum clique in an induced subgraph $G(R)$ is denoted by $\omega(R)$.

3. Efficient algorithms for finding a maximum clique

3.1 Basic Algorithms

3.1.0 A basic branch-and-bound 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 and Q_{max}, where $Q = \{p_1, p_2, ..., p_d\}$ consists of vertices of a current clique and Q_{max} consists of vertices of the largest clique found so far. Let $R = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \cdots \cap \Gamma(p_d) \subseteq V$ consist of *candidate* vertices that can 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

```
procedure Algorithm #0 ($G = (V, E)$)
begin
    global $Q := \emptyset$;   global $Q_{max} := \emptyset$;
    EXPAND($V$);
    output $Q_{max}$
end {of Algorithm #0}

procedure EXPAND($R$)
begin
    while $R \neq \emptyset$ do
        $p$ := a vertex in $R$;        { a vertex for expansion }
        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. 1. Algorithm #0

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 $R_p \neq \emptyset$.

Here, **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 and is shown in Fig.1. We call it **Algorithm #0** [13] and it serves as a reference algorithm. This process can be represented by a *search tree* with root V; whenever $R_p := R \cap \Gamma(p)$ is applied, then R_p is a child of R.

3.1.1 Ordering of vertices If the vertices are sorted in an ascending order with respect to their degrees prior to the application of Algorithm #0 and the vertices are expanded in this order, then the above Algorithm #0 is more efficient. This fact was experimentally confirmed in [13]. Algorithm #0 combined with this vertex-ordering preprocessing is named **Algorithm #1** [13].

Carraghan and Pardalos [11] also employed a similar technique successfully.

3.1.2 Pruning by approximate coloring: *Numbering* One of the most important points for improving the efficiency of the basic Algorithm #0 is to strengthen the *bounding condition* to prune unnecessary searches.

For a set R of vertices, let $\chi(R)$ be the chromatic number of R, *i.e.*, the *minimum* number of colors such 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 such that all pairs of adjacent vertices are colored by different colors. Then we have $\omega(R) \leq \chi(R) \leq \chi'(R) \leq |R|$. An appropriate chromatic number $\chi'(R)$ could be a better upper bound on $\omega(R)$ than $|R|$, and might be obtained with low overhead. Here, we employ a very simple *greedy* or *sequential* approximate coloring to the vertices of R, as introduced in [41]. Let positive integral numbers 1, 2, 3, ... stand for different colors. *Coloring* is also called *Numbering.* For each vertex $q \in R$, *sequentially* from the first to the last, we assign a positive integral *Number* $No[q]$ which is as small as possible. That is, for the vertices in $R = \{q_1, q_2, \ldots, q_m\}$, first let $No[q_1] = 1$, and subsequently, let $No[q_2] = 2$ if $q_2 \in \Gamma(q_1)$ else $No[q_1] = 1, \ldots$, and so on.

We select p (at the 4th line in **procedure** EXPAND(R) in Fig. 1) so that $No[p] = \mathrm{Max}\{No[q] \mid q \in R\}$, where $No[p]$ is an approximate chromatic number of R. Thus, we modify the basic bounding condition:

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

in Fig. 1 Algorithm #0, to the following *new bounding condition*:

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

In addition, to make the above bounding condition more effective, we sort the vertices in R in descending order with respect to their degrees prior to *Numbering*.

We have now an improved algorithm, named **Algorithm #2** [13], as follows: First, sort the vertices in $R = V$ as above. Second, give $Numbers$ to the sorted vertices in $R = V$. Subsequently, apply the modified Algorithm #0 as described above. Note that the sorting and $Numbering$ are applied only once prior to the first application of EXPAND() at depth 0 of the search tree and that the $Numbers$ are inherited in the following EXPAND(). In general, Algorithm #2 is more efficient than Algorithm #1 [13].

We can reduce the search space more effectively by applying the sorting and $Numbering$ of vertices prior to every application of EXPAND(), but with the potential for more overhead and thus, more overall computing time. We confirmed that *adaptive control* of the application of sorting and/or $Numbering$ is effective in reducing the overall computing time [37, 29, 21]. By restricting the application of vertex sorting, as in Algorithm #2, but applying the $Numbering$ to vertices prior to every EXPAND(), we obtain another efficient algorithm, **MCLIQ** [43].

3.2 Algorithms MCQ, MCR, MCS, and MCT

3.2.1 Algorithm MCQ The algorithm MCQ [45] is directly improved from MCLIQ. At the beginning of MCQ, vertices are sorted in descending order with respect to their degrees. Subsequently, we apply $Numbering$ and sorting prior to each EXPAND() operation, where vertices are sorted in an ascending order with respect to their $Numbers$. Then, the last vertex with the *maximum Number* is expanded step-by-step. This sorting can be carried out with little overhead. Hence, MCQ is very simple and efficient.

3.2.2 Algorithm MCR Algorithm MCR [47] is an improved version of MCQ, where the improvements mainly address the *initial* vertex sorting. 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 $1 \leq i \leq |V|$ as in [11]. Here, the degrees of adjacent vertices are also considered. In addition, vertices are assigned initial $Numbers$. This improvement is described precisely in the steps from {SORT} to just above EXPAND(V, No) in Fig.4 (Algorithm MCR) in [47], called *EXTENDED INITIAL SORT-NUMBER* to V.

3.2.3 Algorithm MCS Algorithm MCS [39, 48, 50] is a further improved version of MCR that introduces the following techniques:

(1) Re-NUMBER Because of the *bounding condition* mentioned above, if $No[r] = \mathrm{Max}\{No[q] \mid q \in R\} \leq |Q_{max}| - |Q|$ then it is not necessary to search from vertex r. Let $No_{th} := |Q_{max}| - |Q|$. When we encounter a vertex p with $No[p] > No_{th}$, we attempt to change its $Number$ as follows: Try to find a vertex q in $\Gamma(p)$ such that $No[q] = k_1 \leq No_{th} - 1$, with $|C_{k_1}| = 1$. If such q is found, then try to find $Number\ k_2$ such that no vertex in $\Gamma(q)$ has $Number\ k_2$. If such $Number\ k_2$ is found, then exchange the $Numbers$ of q and p so that $No[q] = k_2$ and $No[p] = k_1$. When this is possible, *it is no longer necessary to search from p*. See Fig.2 for an illustration.

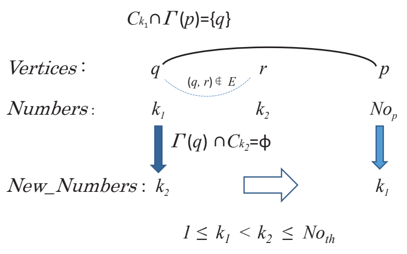
$C_{k_1} \cap \Gamma(p) = \{q\}$
Vertices :
q
r
p
$(q, r) \notin E$
Numbers :
k_1
k_2
No_p
$\Gamma(q) \cap C_{k_2} = \phi$
New_Numbers :
k_2
k_1
$1 \leq k_1 < k_2 \leq No_{th}$

Fig. 2. Re-NUMBER

The above procedure is named Re-NUMBER to p and is very effective.

(2) Adjunct ordered set of vertices for approximate coloring The ordering of vertices plays an important role in the algorithms as described in Sections 3.1.1 and 3.1.2. 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 the first stage of MCR [47]. However, the vertices are *disordered* in succeeding stages, owing to the application of Re-NUMBER. 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.

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 p for expansion in R in which vertices are sorted in ascending order with respect to their $Numbers$ as in MCQ and MCR, for *searching* from the last vertex with the *maximum Number*. Finally, we reconstruct the adjacency matrix in MCR just after the EXTENDED INITIAL SORT-NUMBER to establish a more effective use of the cache memory.

The individual contributions of the above techniques in MCS can be found in Tables 2–4 in [50].

3.2.4 Algorithm MCT An improved algorithm MCT [15, 52] is obtained by modifying MCS in the following ways:

(1) An approximate solution as an initial lower bound We turn back to our original MCS [39] that initially employs an approximation algorithm, init-lb, for the maximum clique problem, to obtain an <u>init</u>ial <u>l</u>ower <u>b</u>ound on the size of the maximum clique. When a sufficiently large near-maximum clique Q'_{max} is found, we let $Q_{max} := Q'_{max}$ at the beginning of MCS [39]. Then $No_{th} := |Q_{max}| - |Q|$ becomes large and the bounding condition becomes more effective. Our init-lb is a local search algorithm based on our previous work [44]. Here, we choose another approximation algorithm, called *k-opt local search* (KLS) [19], by Katayama et al. Recently, Batsyn et al. [6] and Maslov et al. [25] also demonstrated the effectiveness of an approximate solution, independently.

(2) Adaptive application of the sorting and/or *Numbering* of vertices The effectiveness of this approach was already confirmed, as described at the end of Section 3.1.2 [37, 29, 21]. We modify MCS so that we sort the set of vertices by the EXTENDED INITIAL SORT-NUMBER at the first stage near and including the root of the search tree. Konc and

Table 1. CPU time [sec] for benchmark graphs

Graph				KLS [19]		MCR [47]	MCS [48]	MCT [52]	MCX [36]	MaxC [23]	I&M [25]	BG14 [6]
Name	n	$d.$	ω	*sol*	$t.$							
brock400_1	400	0.75	27	25	0.1	729	288	**116**	150	205	188	302
brock800_1	800	0.65	23	21	0.2	7,582	4,122	**1,950**	2,690	4,560	4,000	4,220
brock800_4	800	0.65	26	20	0.2	3,248	1,768	**819**	1,100	1,850	1,680	1,870
C250.9	250	0.90	44	44	0.1	15,386	1,171	404	713	**268**		
gen400_p0.9_65	400	0.90	65	65	0.3	$> 6 \times 10^6$	57,385	**0.74**	66,100	36,700	2,130	19
gen400_p0.9_75	400	0.90	75	75	0.3	$> 3 \times 10^6$	108,298	**0.33**	47,200	9,980	84	7.8
MANN_a45	1035	0.99	345	344	22	653.3	53.4	75.5	32.0	22.7	**17.3**	55.1
p_hat700-3	700	0.75	62	62	0.5	25,631	900	**216**	680	879	552	767
p_hat1000-3	1000	0.74	68	68	1.0	$> 2 \times 10^6$	305,146	**38,800**				
p_hat1500-1	1500	0.25	12	11	0.0	2.22	1.82	**1.40**	1.95	10.00	478	422
p_hat1500-2	1500	0.51	65	65	0.7	268,951	6,299	**1,560**	3,850	8,030	5,350	5,430
san1000	1000	0.50	15	10	0.1	2.16	1.02	**0.21**	0.68	0.72	449	158
sanr400_0.7	400	0.70	21	21	0.1	158.7	77.3	**40.7**	44.5	81.2	86.2	81.4
DSJC500.5	500	0.50	13	13	0.0	1.9	1.5	1.2	**0.8**	2.8		
DSJC1000.5	1000	0.50	15	15	0.1	182	141	**93**	102	265		
keller5	776	0.75	27	27	0.3	45,236	82,421	10,000	30,300	**4,980**	5,780	82,500
r200.8	200	0.80	24-27	24-27	0.0	4.56	1.66	**0.78**	0.95	1.08		
r200.95	200	0.95	58-66	58-66	0.1	218.2	21.1	10.3	30.2	**2.5**		
r300.8	300	0.80	28-29	28-29	0.1	528	161	**61**	89	87		
r400.7	400	0.70	21-22	20-22	0.1	150.1	73.9	**34.9**				
r500.6	500	0.60	17-18	16-17	0.1	27.1	18.0	11.4	**10.1**	22.1		
r500.7	500	0.70	22-23	21-22	0.1	1,533	723	**340**	423	564		
r1000.5	1000	0.50	15-16	14-15	0.1	177	134	**92**	103	231		
r2000.4	2000	0.40	13-14	12	0.2	548	460	**366**				
r3000.2	3000	0.20	9	7-8	0.1	3.94	3.67	**3.42**	4.34	34.40		
r3000.3	3000	0.30	11	10-11	0.2	138	121	**107**				
r3000.4	3000	0.40	14	12-13	0.5	7,834	6,392	**5,152**				
r5000.2	5000	0.20	9-10	7-8	0.2	46.8	44.6	**39.0**	69	578		
r5000.3	5000	0.30	12	10-11	0.5	2,636	2,284	**1,875**				
r10000.1	10000	0.10	7	5-6	0.6	15	**14**	**14**	20	684		
r10000.2	10000	0.20	10	8-9	0.9	1,475	1,303	**1,139**				
r15000.1	15000	0.10	8	6	1.3	80	**62**	**62**	115	2,749		
r20000.1	20000	0.10	8	6-7	2.3	307	**234**	**234**				

Janežič [22] were also successful in improving MCQ in a similar way as in [21], independently.

In contrast, mainly near the leaves of the search tree, to lighten the overhead of preprocessing before expansion of vertices, we only inherit the order of vertices from that in their parent depth, and we merely inherit the $Numbers$ from those assigned in their parent depth if their $Numbers$ are less than or equal to No_{th}. For vertices whose inherited $Numbers$ are greater than No_{th}, we give them new $Numbers$ by sequential $Numbering$ combined with $Re - Numbering$.

Table 1 shows the progression of the running times required to solve some benchmark graphs using the above algorithms within these ten years [52]. Here, $d.$ indicates the *density* of the graph, and *sol* and $t.$ show the solution and the computing time of KLS in MCT. In the last half of the table, r$n.p$ stands for a random graph, with the number of vertices $= n$ and the edge probability $= p$. The results of the state-of-the-art algorithm BBMCX (MCX *for short*) [36] by Segundo et al. and of other algorithms [23, 25, 6] are also included for reference [52]. Note that MaxCLQ (MaxC *for short*) [23] by Li and Quan is fast for dense graphs. ILS&MCS (I&M *for short*) [25] and BG14 [6] require more time than MCT for most of the instances tested. One reason for this difference comes from the fact that our approximation algorithm, KLS, takes only small portion of the whole algorithm's computing time, whereas their approximation algorithm, ILS, [2] in I&M and BG14 consumes a considerable part of the whole computing time.

4. Efficient algorithm for enumerating all maximal cliques

In addition to finding one maximum clique, enumerating all maximal cliques is also important and has diverse applications. We present a depth-first search algorithm, **CLIQUES** [42, 46], for enumerating all maximal cliques of an undirected graph $G = (V, E)$. All maximal cliques enumerated are output in a tree-like form. The basic framework of CLIQUES is almost the same as that of Algorithm #0 *without the basic bounding condition.* We maintain a global variable $Q = \{p_1, p_2, ..., p_d\}$ that consists of the vertices of a current clique, and let $SUBG = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \cdots \cap \Gamma(p_d)$. We begin the algorithm by letting $Q := \emptyset$ and $SUBG := V$ (the set of all vertices). We select a certain vertex p from $SUBG$ and add p to Q ($Q := Q \cup \{p\}$). Then, we compute $SUBG_p := SUBG \cap \Gamma(p)$ as the new set of *candidate* vertices. In particular, the initially selected vertex $u \in SUBG$ is called a *pivot.* This EXPAND() procedure is applied recursively while $SUBG_p \neq \emptyset$.

We describe two methods to prune unnecessary parts of the search tree, which happen to be the same as in the Bron-Kerbosch algorithm [8]. We regard the set $SUBG$ ($= V$ at the beginning) as an *ordered* set of

```
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 $| CAND \cap \Gamma(u) |$;{pivot}
 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. 3. Algorithm CLIQUES

vertices, and we continue to enumerate maximal cliques from vertices in $SUBG$ step-by-step 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 *fini*shed). Then we denote by $CAND$ the set of remaining candidates for expansion: $CAND = SUBG - FINI$. Initially, $FINI := \emptyset$ and $CAND := SUBG$. In the subgraph $G(SUBG_q)$ with $SUBG_q := SUBG \cap \Gamma(q)$, let

$$FINI_q := SUBG_q \cap FINI,$$
$$CAND_q := SUBG_q - FINI_q.$$

Then only the vertices in $CAND_q$ can be candidates for expanding the clique $Q \cup \{q\}$ to find *new* larger cliques.

Second, for the initially selected pivot u in $SUBG$, any maximal clique Q' in $G(SUBG \cap \Gamma(u))$ is not maximal in $G(SUBG)$, since $Q' \cup \{u\}$ is a larger clique in $G(SUBG)$. Therefore, searching for maximal cliques from $SUBG \cap \Gamma(u)$ should be excluded.

Taking the previously described pruning method into consideration, the only search subtrees to be expanded are from vertices in $(SUBG - SUBG \cap \Gamma(u)) - FINI = CAND - \Gamma(u)$. Here, to minimize $|CAND - \Gamma(u)|$, we choose the pivot $u \in SUBG$ that *maximizes* $|CAND \cap \Gamma(u)|$, which is *crucial* to establish the *optimality* of the worst-case time-complexity of the algorithm. This kind of pivoting strategy was first proposed by Tomita et al. [42].

The algorithm CLIQUES [42, 46] is shown in Fig.3, which enumerates all maximal cliques based upon the above approach, where all maximal cliques enumerated are presented in a tree-like form. Here, if Q is a *maximal* clique that is found at statement 2, then the algorithm only prints out the string of characters "*clique*," instead of Q itself at statement 3. Otherwise, it is impossible to achieve the optimal worst-case running time. 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 the 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 tree-like output format is also important *practically*, since it saves space in the output file.

We have proved that the worst-case time-complexity of CLIQUES is $O(3^{n/3})$ for an n-vertex graph [42, 46]. This is *optimal* as a function of n, since there exist up to $3^{n/3}$ cliques in an n-vertex graph [27]. The algorithm was also demonstrated to run fast in practice through computational experiments [46]. An example run of CLIQUES can be found in [51] together with those of [53] by Tsukiyama et al. and [24] by Makino-Uno applied to the same graph. By combining a bounding rule with CLIQUES, we obtained a simple $O(2^{n/2.863})$-time algorithm, **MAXCLIQUE** [38], for finding a maximum clique. It was experimentally shown in [38] that MAXCLIQUE runs faster than Tarjan and Trojanowsky [40]'s $O(2^{n/3})$-time algorithm.

In this approach, Eppstein et al. [12] proposed an algorithm for enumerating all maximal cliques that runs in time $O(dn3^{d/3})$ for an n-vertex graph G, where d is the *degeneracy* of G which is defined to be the smallest number such that every subgraph of G contains a vertex of degree at most d. If the graph G is *sparse*, d can be much smaller than n; hence $O(dn3^{d/3})$ can be much smaller than $O(3^{n/3})$.

Exact cliques are often too restrictive for practical applications as has been pointed in [35]. A useful algorithm for enumerating pseudo-cliques in large scale networks (graphs) has recently been proposed [57].

5. Applications

Many applications of maximum and maximal cliques can be found in [34, 7, 10, 49, 54], and others. Thefore, we refer only to some of the literature in the following fields:

(a) Boinformatics
  a-1) Analysis of protein structures [3–5, 1, 9]
  a-2) Analysis of glycan structures [14, 28]
(b) Data mining
  b-1) Basic algorithms
    • Structural change pattern mining [31, 32]
    • Pseudo clique enumeration [56, 33, 57]
  b-2) Practical applications
    • Data mining for related genes [26]
    • Structural analysis of enterprise relationship [55]

(c) Image processing
- Face detection [16]

(d) Design of quantum circuits [30]
(e) Design of DNA and RNA sequences for bio-molecular computation [20]

Acknowledgments

The author would like to express his sincere gratitude to H. Ito, T. Akutsu, M. Haraguchi, Y. Okubo, T. Nishino, H. Takahashi and many others for their fruitful joint work and kind help. This work was supported by JSPS KAKENHI Grant Numbers JP16300001, JP19300040, JP19500010, JP21300047, JP22500009, JP25330009, Kayamori Foundation of Informational Science Advancement, Funai Foundation for Information Technologies, and others.

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. Fundamentals of Electronics, Communications and Computer Sciences, E89-A, 1215-1222 (2006).
2. Andrade, D.V., Resende, M.G.C., Werneck, R.F.: Fast local search for the maximum independent set problem, J. Heuristics, 18, 525–547 (2012)
3. 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)
4. 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, 103-126 (2005)
5. 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)
6. Batsyn, M., Goldengorin, B., Maslov, E., Pardalos, P. M.: Improvements to MCS algorithm for the maximum clique problem, J. Comb. Optim., 27, 397-246 (2014)
7. 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)
8. Bron, C., Kerbosch, J.: Algorithm 457, Finding all cliques of an undirected graph, Comm. ACM, 16, 575–577 (1973)
9. 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, 3–12 (2006)
10. Butenko, S., Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics - Invited Review - , European J. Operational Research, 173, 1–17 (2006)
11. Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem, Operations Research Lett., 9, 375–382 (1990)
12. Eppstein, D., Löffler, M., Strash, D.: Listing all maximal cliques in large sparse real-world graphs, J. Experimental Algorithmics, 18, 3.1:1–21 (2013)
13. Fujii, T., Tomita, E.: On efficient algorithms for finding a maximum clique, Tech. Rep. IECE, AL81-113, 25–34 (1982)
14. 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, 12(S-1)S:13 (2011)
15. Hatta, T., Tomita, E., Ito, H., Wakatsuki, M.: An improved branch-and-bound algorithm for finding a maximum clique, Proc. Summer LA Symposium, No.9, 1–8 (2015)

16. 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)
17. Johnson, D.S., Trick, M.A. (Eds.): Cliques, Coloring, and Satisfiability, DIMACS Series in Discrete Math. and Theoret. Comput. Sci., vol.26, American Math. Society (1996)
18. 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)
19. Katayama, K., Hamamoto, A., Narihisa, H.: An effective local search for the maximum clique problem, Information Processing Lett., 95, 503-511 (2005)
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, Tech. Rep. IEICE, COMP89-113, 1–8 (1990)
22. Konc, J., Janežič, D.: An improved branch and bound algorithm for the maximum clique problem, MATCH Commun. Math. Comput. Chem., 58, 569–590 (2007)
23. Li, C.M., Quan, Z.,: Combining graph structure exploitation and propositional reasoning for the maximum clique problem, Proc. IEEE ICTAI, 344–351 (2010)
24. Makino, K., Uno, T.: New algorithms for enumerating all maximal cliques, SWAT 2004, LNCS, 3111, 260–272 (2004)
25. Maslov, E., Batsyn, M., Pardalos, P.M.: Speeding up branch and bound algorithms for solving the maximum clique problem, J. Global Optim., 59, 1-21 (2014)
26. Matsunaga, T., Yonemori, C., Tomita, E., Muramatsu, M.: Clique-based data mining for related genes in a biomedical database, BMC Bioinformatics, 10:205 (2009)
27. Moon, J.W., Moser, L.: On cliques in graphs, Israel J. Math., 3, 23–28 (1965)
28. Mori, T., Tamura, T., Fukagawa, D., Takasu, A., Tomita, E., Akutsu, T.: A clique-based method using dynamic programming for computing edit distance between unordered trees, J. Computational Biology, 19, 1089-1104 (2012)
29. Nagai, M., Tabuchi, T., Tomita, E., Takahashi, H.: An experimental evaluation of some algorithms for finding a maximum clique, Conf. Records of the National Convention of IEICE 1988, D-348 (1988)
30. 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. Tech. Rep. RIMS, 1325, Kyoto University, 45–50 (2003)
31. Okubo, Y., Haraguchi, M., Tomita, E.: Structural change pattern mining based on constrained maximal k-Plex search, DS 2012, LNAI, 7569, 284–298 (2012)
32. Okubo, Y., Haraguchi, M., Tomita, E.: Relational change pattern mining based on modularity difference, MIWAI 2013, LNAI, 8271, 187-198 (2013)
33. Okubo, Y., Haraguchi, M., Tomita, E.: Enumerating maximal isolated cliques based on vertex-dependent connection lower bound, MLDM 2016, LNAI, 9727, 569–583 (2016)
34. Pardalos, P.M. and Xue, J.: The maximum clique problem, J. Global Optim., 4, 301–328 (1994)
35. Pattillo, J., Youssef, N. Butenko, S.: Clique relaxation models in social network analysis, Thai, M. T., Pardalos, P. M. (eds.): Handbook of Optimization in Complex Networks: Communication and Social Networks, Springer Optimization and Its Applications 58, 143–162 (2012)
36. Segundo, P.S., Nikolaev, A., Batsyn, M.: Infra-chromatic bound for exact maximum clique search, Computers and Operations Research, 64, 293–303 (2015)
37. Shindo, M., Tomita, E., Maruyama, Y.: An efficient algorithm for finding a maximum clique, Tech. Rep. IEC, CAS86-5, 33–40 (1986)
38. Shindo, M., Tomita, E.: A simple algorithm for finding a maximum clique and its worst-case time complexity, Systems and Computers in Japan, 21, Wiley, 1-13 (1990)

39. Sutani, Y., Higashi, T., Tomita, E. Takahashi, S., Nakatani, H.: A faster branch-and-bound algorithm for finding a maximum clique, Tech. Rep. IPSJ, 2006-AL-108, 79–86 (2006)
40. Tarjan, R.E., Trojanowski, A.E.: Finding a maximum independent set, SIAM J. Computing, 6(3), 537–546 (1977)
41. Tomita, E., Yamada, M.: An algorithm for finding a maximum complete subgraph, Conf. Records of the National Convention of IECE 1978, 8 (1978)
42. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for finding all the cliques, Tech. Rep. Univ. of Electro-Commun., UEC-TR-C5(2) (1988). (Reference [238] in [34], Reference [308] in [7]).
http://id.nii.ac.jp/1438/00001898/
43. Tomita, E., Kohata, Y., Takahashi, H.: A simple algorithm for finding a maximum clique, Tech. Rep. Univ. of Electro-Commun., UEC-TR-C5(1) (1988). (Reference [239] in [34], Reference [309] in [7]).
http://id.nii.ac.jp/1438/00001899/
44. Tomita, E., Mitsuma, S., Takahashi, H.: Two algorithms for finding a near-maximum clique, Tech. Rep. Univ. of Electro-Commun., UEC-TR-C1 (1988). (Reference [240] in [34], Reference [310] in [7]).
http://id.nii.ac.jp/1438/00001900/
45. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique, DMTCS 2003, LNCS, 2731, 278–289 (2003)
46. Tomita, E., Tanaka, A. Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments, Theoret. Comput. Sci., 363 (Special Issue on COCOON 2004), 28–42 (2006)
47. 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)
48. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique, WALCOM 2010, LNCS, 5942, 191-203 (2010)
49. Tomita, E., Akutsu, T., Matsunaga, T.: Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics, In: Laskovski, A.N. (Ed.): Biomedical Engineering, Trends in Electronics, Communications and Software, InTech, 625-640 (2011)
http://cdn.intechopen.com/pdfs-wm/12929.pdf
50. Tomita, E., Sutani, Y., Higashi, T., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments, IEICE Trans. Information and Systems, E96-D, 1286-1298 (2013).
http://id.nii.ac.jp/1438/00000287/
51. Tomita, E.: Clique Enumeration, in Ming-Yang Kao (Ed.): Encyclopedia of Algorithms, 2nd Edition, Springer, 313-317 (2016)
52. Tomita, E., Yoshida, K., Hatta, T., Nagao, A., Ito, H., Wakatsuki, M.: A much faster branch-and-bound algorithm for finding a maximum clique, FAW 2016, LNCS, 9711, 215–226 (2016)
53. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all the maximal independent sets, SIAM J. Comput., 6, 505–517 (1977)
54. Wu, Q., Hao, J.K.: A review on algorithms for maximum clique problems - Invited Review - , European J. Operational Research, 242, 693–709 (2015)
55. Yonemori, C., Matsunaga, T., Sekine, J., Tomita, E.: A structural analysis of enterprise relationship using cliques, DBSJ Journal, 7, 55-60 (2009)
56. Zhai, H., Haraguchi, M., Okubo, Y., Tomita, E.: Enumerating maximal clique sets with pseudo-clique constraint, DS 2015, LNAI, 9356, 324–339 (2015)
57. Zhai, H., Haraguchi, M., Okubo, Y., Tomita, E.: A fast and complete algorithm for enumerating pseudo-cliques in large graphs, Int. J. Data Science and Analytics, 2, Springer, 145–158 (2016)