アルゴリズム 12－13
(1989.11.21)

An Optimal Algorithm for Finding All the Cliques*

Etsuji Tomita Akira Tanaka** Haruhisa Takahashi

Department of Communications and Systems
The University of Electro-Communications
Chofugaoka, Chofu, Tokyo 182 Japan

Abstract

We present an algorithm for finding all the cliques of an undirected graph. It is a very simple backtracking searching algorithm, whose basic techniques are based upon Bron and Kerbosch's, and it outputs the cliques found in a tree-like form. Then we prove that its worst-case time complexity is $O(3^{n/3}) = O(2^{n/1.89\ldots}) = O(1.44\ldots^{n})$ for an n-vertex graph. This is optimal *with respect to n* since there exist up to $3^{n/3}$ cliques in an n-vertex graph. It is noted that a slight modification of this algorithm gives an $O(2^{n/2.98})$-time simple algorithm for finding *only one* maximum clique.

1. Introduction.

A maximal complete subgraph of an undirected graph G is called a *clique*. A set of vertices of a clique of the complementary graph $\overline{G}$ is a *maximal independent set* of G. Finding all the cliques or all the maximal independent sets of a given graph is a fundamental problem in the theory of graphs and has many diverse applications.

Then a number of algorithms have been presented and evaluated experimentally or theoretically. Among them, Tsukiyama et al.[TIAS] presented an algorithm for generating all the maximal independent sets in a graph G in $O(nm)$ time per maximal independent set, where n and m are the numbers of vertices and edges of G, respectively. Furthermore, Lawler et al. [LLK] generalized its result. In addition, Chiba and Nishizeki[CN] improved Tsukiyama et al.'s algorithm much to have a more efficient one for listing all the cliques of G in $O(a(G)m)$ per clique, where $a(G)$ is the arboricity of G with $a(G) \leqq O(m^{1/2})$ for a connected graph G. However, any nontrivial time complexity analysis was ever

*This work was partially supported by Grants-in-Aid for Scientific Research Nos. 60550259 and 62550259 from the Ministry of Education, Science and Culture, Japan.

**Presently with TOYOTA Motor Corporation.

given for no algorithms with respect to n, the number of vertices, except those given experimentally [BK], [J], [RND,p.390]. The time complexity analysis of this kind is especially of interest when it is compared with the result by Tarjan and Trojanowski [TT] who gave an $O(2^{n/3})$-time algorithm for finding *only one* maximum independent set, or that by Robson [R] who showed an $O(2^{0.276n})$-time algorithm (which uses exponential space) for the same problem.

Now we present here an algorithm for finding all the cliques of an undirected graph whose basic techniques are based upon Bron and Kerbosch[BK]. Then we prove that its worst-case running time complexity is $O(3^{n/3})=O(2^{n/1.89\ldots})$ for a graph with n vertices. This is the optimal result with respect to n, since there exist up to $3^{n/3}=3^k$ cliques in a graph with $n=3k$ vertices as shown by Moon and Moser[MM].

At the end of this report, we also present some of our new results which are relevant to this problem.

2. Preliminaries.

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

[2] For a vertex $v\in V$, let $\Gamma(v)$ be the set of all vertices which are adjacent to v in $G=(V,E)$, i.e.,

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

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

[4] Given a subset $Q\subseteq V$ of vertices, the induced subgraph $G(Q)$ is said to be *complete* if $(v,w)\in E$ for all $v, w\in Q$ with $v\neq w$. If this is the case, we may simply say that Q is a complete subgraph. In particular, if a complete subgraph is *maximal*, then it is called a *clique*. A subset $R\subseteq V$ of vertices is said to be *independent* if $(v,w)\notin E$ for all $v,w\in R$. Here, $Q\subseteq V$ is a clique of G if and only if Q is a maximal independent set of the complementary graph $\overline{G}$.

3. The algorithm.

We present a backtracking searching algorithm CLIQUES for finding all the cliques of a given graph $G=(V,E)$ $(V\neq\phi)$ that is essentially based upon Bron and Kerbosch[BK].

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

Let $Q=\{p_1, p_2, \cdots, p_d\}$ be found

to be a complete subgraph at some stage, and consider a *subgraph* G(SUBG) which is induced by a set of vertices

SUBG

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

where SUBG=V when $Q=\phi$ at the initial stage. Then apply the procedure EXTEND to SUBG searching for larger complete subgraphs. If $SUBG=\phi$ then Q is clearly a *maximal* complete subgraph, i.e., a clique. Otherwise, $Q\cup\{q\}$ is a larger complete subgraph for every $q\in$ SUBG. Then consider smaller subgraphs $G(SUBG_q)$ which are induced by new sets of vertices

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

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

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

Now we proceed to describe two methods to prune unnecessary parts of the search forest.

First, for the previously described set $SUBG(\neq\phi)$, let

$$SUBG=FINI\cup CAND$$

$$(FINI\cap CAND=\phi).$$

Here suppose that we have already *finished* extending search subtrees from every vertex $q'\in FINI\subseteq SUBG$ to find all the cliques containing $Q\cup\{q'\}$, and that only the remaining vertex $q\in CAND\subseteq SUBG$ is a *candidate* for further extension of the present complete subgraph Q to find new cliques. Consider the subgraph $G(SUBG_q)$ with $SUBG_q=SUBG\cap \Gamma(q)$, and let

$$SUBG_q=FINI_q\cup CAND_q$$

$$(FINI_q\cap CAND_q=\phi),$$

where

$FINI_q=FINI\cap \Gamma(q)$, and

$CAND_q=CAND\cap \Gamma(q)$.

Then only the vertices in the subgraph $G(CAND_q)$ can be candidates for extending the complete subgraph $Q\cup\{q\}$ to find *new* larger cliques. Thus, further extension is to be considered only for vertices in $G(CAND_q)$ excluding ones in $FINI_q=SUBG_q-CAND_q$.

Secondly, given a certain vertex $u\in$ SUBG, suppose that all the cliques containing $Q\cup\{u\}$ have been found. Then every *new* clique containing Q, but not $Q\cup\{u\}$, must contain some vertex $q\in SUBG-\Gamma(u)$. This is because if Q is extended to a complete subgraph $R=Q\cup V$ with $V\subseteq \Gamma(u)$ and $u\notin R$, then $R\cup\{u\}$ is a *larger* complete subgraph, so R is not maximal. Thus, any new clique can be found by extending Q to $Q\cup\{q\}$, where $q\in SUBG-\Gamma(u)$, and finding all the cliques containing $Q\cup\{q\}$. Therefore, if we extend a search subtree from u, further extension is to be considered only for vertices in $SUBG-\Gamma(u)$. Taking the previous pruning method into consideration, too, the only search

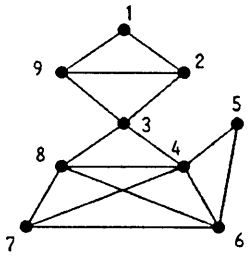
subtrees to be extended are from vertices in (SUBG-Γ(u))-FINI = CAND-Γ(u) ($\ni$u). Here, in order to minimize |CAND-Γ(u)|, we choose such a vertex u$\in$SUBG to be the one which maximizes |CAND $\cap$ Γ(u)|. In this way, the problem of finding all the cliques of G(CAND) can be decomposed into k=|CAND-Γ(u)| such subproblems, see LEMMA (i) below.

With these two pruning methods, we have the algorithm CLIQUES for finding all the cliques without duplications as shown below.

Here, every time Q is found to be a clique at statement 2, we only print out a string of characters "clique," instead of Q itself in statement 3. This is because, otherwise, it is impossible to achieve the worst-case running time of $O(3^{n/3})$ for an n-vertex graph, since printing out of Q requires time proportional to the size of Q which is a global variable. Instead, in addition to statement 3, not only we print out q followed by a comma at statement 7 every time q is picked out as a

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

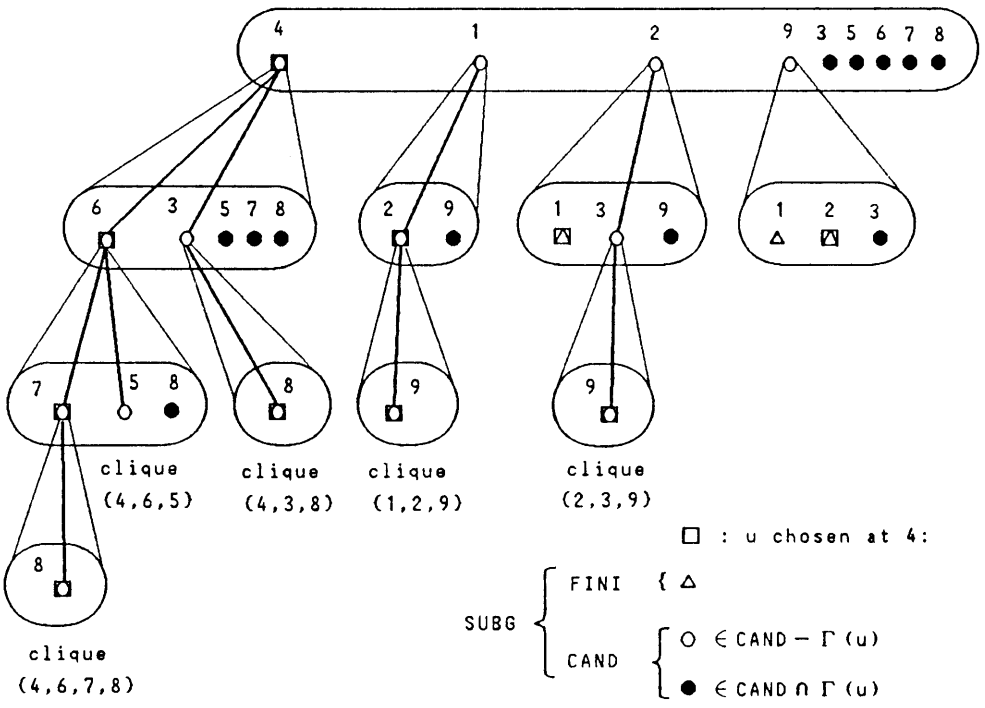
new element of a complete subgraph, but also we print out a string of characters "back," at statement 11 after q is moved from CAND to FINI at statement 10. We can easily obtain a tree representation of all the cliques from the resultant sequence printed by statements 3, 7, and 11. This transformation can be done in time proportional to the length of the resultant sequence. Here, primed statements 0′, 7′, and 11′ are only for the sake of explanation, and should be deleted finally.



(a) An input graph G

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

(c) A resultant printed sequence



(b) A search forest for G

FIG.1. An example

Example. Let us apply the above algorithm CLIQUES to a graph in FIG.1(a). Then the whole process is represented by a seach forest in FIG.1(b), and we have the resultant printed sequence in FIG.1(c). In FIG.1(b), each set of vertices surrounded by a flat circle represents SUBG at the stage, in which vertex with △ mark is in FINI⊆ SUBG at the beggining . Vertex u chosen in statement 4 is marked by ◻ or ◬ depending on whether it is in CAND or FINI, respectivily. Other vertices in CAND-Γ (u) are marked by ○, while vertices in CAND∩ Γ (u) are marked by ●. Thus, all the cliques of G are {4,6,7,8}, {4,5,6}, {4,3,8}, {1,2,9}, and {2,3,9}.

4. The worst-case time complexity.

We evaluate the worst-case running time of the previous algorithm CLIQUES(G) with the primed statements 0′, 7′, and 11′ having been deleted. So, this is equivalent to evaluating that of EXTEND(V, V). Now we begin by giving a few definitions.

[1] Let $T(n,m)$ be an upper bound on the worst-case running time of EXTEND(SUBG, CAND) when $|\,SUBG\,|=n$ and $|\,CAND\,|=m$ ($n\geqq m\geqq 0$).

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

[3] Let us consider a nonrecursive procedure EXTEND$_0$(SUBG, CAND) which is obtained from EXTEND(SUBG, CAND) by deleting a recursive call 9: EXTEND(SUBG$_q$, CAND$_q$). The running time of EXTEND$_0$(SUBG, CAND) when $|\,SUBG\,|=n$ and $|\,CAND\,|=m$ can be made to be $O(n^2)$, then let this running time be less than or equal to the following quadratic equation

$$P(n)=p_1n^2+p_2n+p_3,$$

where $p_1>0$, $p_2\geqq 0$, $p_3\geqq 0$. □

From the above definitions, we have that

$$T(n,m)=\max_{0\leqq k\leqq m}\{T_k(n,m)\}\leqq\max_{1\leqq k\leqq m}\{T_k(n,m)\}$$

since $T_0(n,m)\leqq T_k(n,m)$ for any k, $1\leqq k\leqq m$.

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

LEMMA. Consider EXTEND(SUBG, CAND) when $|\,SUBG\,|=n$, $|\,CAND\,|=m$, $|\,EXT_u\,|=|\,CAND-\Gamma(u)\,|=k\neq 0$, and $|\,CAND\cap\Gamma(u)\,|=\ell$ at the first entrance to statement 5. In what follows, CAND stands only for this initial value, though it is decreased one by one at statement 10 in the **while** loop. Let CAND-$\Gamma(u)=\{v_1, v_2, \cdots, v_k\}$ and the vertex at statement 6 be chosen in this order. Let

$SUBG_i=SUBG\cap\Gamma(v_i)$, and

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

Then

(i) $T_k(|\,SUBG\,|,|\,CAND\,|)\leqq\sum_{i=1}^{k}T(|\,SUBG_i\,|,|\,CAND_i\,|)+P(n)$,

(ii) a) $|\,CAND_i\,|\leqq\ell$, and

b) $|\,SUBG_i\,|\leqq n-k\leqq n-1$.

(See FIG.2) □

THEOREM. The upper bound $T(n,m)$ on the worst-case running time of EXTEND(SUBG, CAND) with |

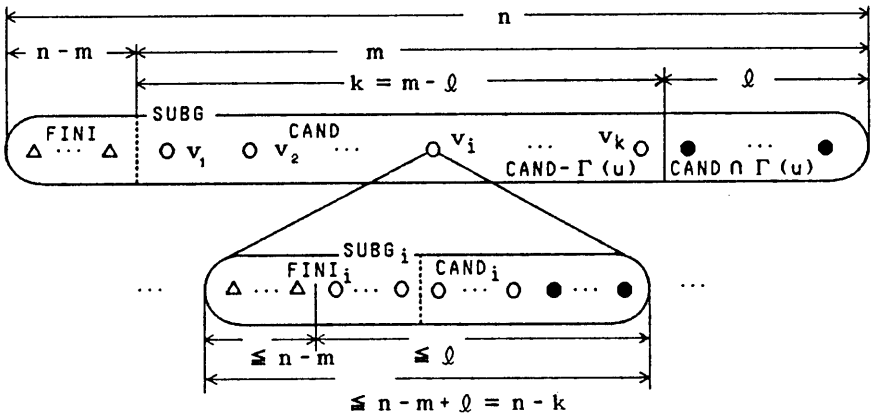


FIG.2. An illustration for LEMMA

SUBG| $=n$ and |CAND| $=m$ is expressed as follows for all $n \geqq m \geqq 0$:

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

where

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

with

$$q_1 = p_1/2 > 0,$$
$$q_2 = (9p_1 + p_2)/2 > 0,$$
$$q_3 = 27p_1/2 + 9p_2/4 + p_3/2 > 0,$$

and

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

with $C_1 = 3q_2/\ln 3$, $C_2 = p_3 + q_3$, and C_3 being the maximum value 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 finite when $n \geqq 1$ is finite and it approaches 0 as n tends to infinity. Hence, C_3 is a finite constant.)

Here, $R(n) \equiv C3^{n/3} - Q(n)$ is monotone increasing with $R(n) \geqq p_3$ for all integers $n \geqq 0$.

Proof. To begin with we can easily prove the monotone increasingness of $R(n)$. Then the preceding main part can be proved by induction on n. □

In particular, since

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

we conclude that the worst-case running time of the algorithm CLIQUES(G) is $O(3^{n/3})$ for an n-vertex graph G=(V,E). Note here that the original Bron and Kerbosch's algorithm outputs the entire clique itself in $O(n)$ time every time it is found. Thus their algorithm takes $O(n3^{n/3})$ time as a whole.

5. Concluding remarks.

It should be noted that the time complexity analysis of the previous algorithm or its variant as a function of the number of cliques of the graph is very hard and is left open.

Now we conclude this report by giving the following notes.

1) A slight modification of algorithm CLIQUES gives an $O(2^{n/2.98})$-time simple algorithm for finding *only one* maximum clique for an n-vertex graph. It is very much simpler than Tarjan and Trojanowski's $O(2^{n/3})$-time algorithm [TT] or Robson's $O(2^{0.276n})$-time algorithm[R] for finding a maximum independent set, while its time complexity analysis is very elaborate. The algorithm has been experimentally confirmed to run faster than Tarjan et al.'s for random graphs with up to 400 vertices.

2) Other modifications with *heuristics* give another algorithm for finding a maximum clique that, in experiment, runs much faster than the previous $O(2^{n/2.98})$-time algorithm. It has been also experimentally confirmed to run faster than Balas and Yu's algorithm[BY] for several graphs.

3) By appropriately restricting the search in the above algorithm, we can obtain an $O(n^3)$-time algorithm for finding a near-maximum clique.

These results will be reported in detail some other time.

Acknowledgments

The authors would like to express their sincere gratitude to Profs. T.Nishizeki, S.Tsukiyama, D. Hochbaum, E.Lawler, R.Karp, J.Reif, and R.A.Wagner who joined in the discussions and gave them useful comments. They also wish to thank M.Shindoh and Y.Miura for their helps.

References

[BK] C.BRON AND J.KERBOSCH, Algorithm 457: Finding all cliques of an undirected graph,Comm.ACM,16(1973),pp.575-577.

[BY] E.BALAS AND C.S.YU, Finding a maximum clique in an arbitrary graph, SIAM J. Comput., 15(1986), pp.1054-1068

[CN] N.CHIBA AND T.NISHIZEKI, Arboricity and subgraph listing algorithms, SIAM J. Comput.,14(1985),pp.210-223.

[J] H.C.JOHNSTON, Cliques of a graph- Variations on the Bron-Kerbosch algorithm, Internat.J.Comput. and Information Sci., 5(1976),pp.209-238.

[LLK] E.L.LAWLER, J.K.LENSTRA AND A.H.G.RINNOOY KAN, Generating all maximal independent sets:NP-hardness and polynomial-time algorithms, SIAM J.Comput.,9(1989),pp.558-565.

[MM] J.W.MOON AND L.MOSER, On cliques in graphs, Israel J.Math., 3(1965)pp.23-28.

[R] J.M.ROBSON, Algorithms for maximum independent sets, J. of Algorithms, 7(1986) pp.425-440.

[RND] E.M.REINGOLD, J.NIEVERGELT, AND N.DEO, Combinatorial Algorithms: Theory and Practice, Prentice-Hall, Englewood Cliffs,NJ, 1977.

[TIAS] S.TSUKIYAMA, M.IDE, H.ARIYOSHI, AND I.SHIRAKAWA, A new algorithm for generating all the maximal independent sets, SIAM J. Comput.,6(1977), pp.505-517.

[TT] R.E.TARJAN AND A.E.TROJANOWSKI, Finding a maximum independent set, SIAM J. Comput.,6(1977), pp.537-546.