Operations Research Letters 9 (1990) 375–382
North-Holland

November 1990

# AN EXACT ALGORITHM FOR THE MAXIMUM CLIQUE PROBLEM

Randy CARRAGHAN

*Computer Science Department, Whitmore Laboratory, USA*

Panos M. PARDALOS

*The Pennsylvania State University, University Park, PA 16802, USA*

Received June 1989
Revised January 1990

A partially enumerative algorithm is presented for the maximum clique problem which is very simple to implement. Computational results for an efficient implementation on an IBM 3090 computer are provided for randomly generated graphs with up to 3000 vertices and over one million edges. Also provided are exact specifications for test problems to facilitate future comparisons. In addition, the Fortran 77 code of the proposed algorithm is given.

maximum clique * enumerative algorithm * test problems * Fortran program

## 1. Introduction

Given a graph $G(V, E)$, where $V$ is the set of vertices and $E$ is the set of edges, a complete subgraph of $G$ is one whose vertices are pairwise adjacent. The *maximum clique problem* is to find the maximum complete subgraph of $G$. An independent set is a set of vertices whose elements are pairwise nonadjacent. The *maximum independent set problem* is to find the largest such indepent set. A vertex cover is a set of vertices that covers all the edges of $G$. The *minimum vertex cover problem* is to find the minimum of such set. These three problems are computationally equivalent. They are also known to be NP-complete (see also [2]).

The maximum clique problem has numerous applications in science and engineering [1]. One way to cope with an NP-complete problem is to design an algorithm that performs fast on the average with respect to a natural probability distribution of inputs. Most of the algorithms for the maximum clique problem degenerate to an exhaustive search. An obvious enumerative algorithm is to consider all subsets of $V$ in decreasing order of cardinality and check if each one forms a clique. Stop the checking when such a set has been found. Algorithms for the maximum clique problem have been studied by many authors (see references [1] and [3]–[8]). Most of the proposed algorithms have been implemented and tested on medium size problems using random graphs. Recently, using a branch and bound method based on quadratic 0–1 programming [7], problems with 1000 vertices and as many as 150 000 edges were solved.

In this paper, we propose a simple and very efficient algorithm for finding the maximum clique in a graph $G$. The algorithm uses a partial enumeration and is very simple to implement. Computational experience with randomly generated graphs shows that the algorithm is faster than any previous known method.

## 2. Algorithm description

Initially, the algorithm considers an ordering of the vertices of $G$, say $v_1, v_2, \dots, v_n$, where $v_1$ is a vertex of smallest degree in $G$, $v_2$ is a vertex of smallest degree in $G - (v_1)$, and generally $v_k$ is a vertex of smallest degree in $G - \{v_1, \dots, v_{k-1}\}$, $k \leqslant n - 2$.

It has been observed that this vertex ordering reduces the computational time in problems with dense graphs. In addition, this ordering can be reapplied at depths $> 1$. For lower density problems the algorithm is faster when no ordering of the vertices is performed.

Without any pruning, the algorithm produces a most $n$ cliques of increasing size. Initially, it finds the largest clique $C_1$ that contains the vertex $v_1$. Then it finds $C_2$, the largest clique in $G - \{v_1\}$ that contains $v_2$ and so on. Applying heuristics and pruning techniques, we can reduce the search space dramatically.

Crucial to the understanding of the algorithm is the notion of the *depth*. Suppose we consider vertex $v_1$. At depth 2, we consider all vertices adjacent to $v_1$. At depth 3, we consider all vertices (that are in depth 2) adjacent to the first vertex in depth 2 (the example below makes this clear). Let $v_{di}$ be the vertex we are currently expanding at depth $d$ and step $i$. That is,

Depth $d$: $v_{d1}, \ldots, v_{di}, \ldots, v_{dm}$.

If $d + (m - i) \leqslant$ size of current best clique (CBC), then we prune, since the size of the largest possible clique (formed by expanding $v_{di}$) would be less or equal to the size of CBC. If we are at depth 1 and this inequality holds then we stop; we have found the maximum clique.

**Remark.** If it is known that the maximum clique has size $\geqslant \alpha$, then we may use as pruning (stopping) criterion the following:

$$d + (m - i) \leqslant \alpha.$$

If $\alpha$ is close to the size of the actual maximum clique, the computational time can be reduced dramatically for graphs with high density. In [9] Turan proved that every graph with $n$ vertices and $m$ edges contains a clique of size $\geqslant n^2/(n^2 - 2m)$. Hence, if the graph has density $D$ then we may take

$$\alpha = \frac{n}{(1 - D)n + D}.$$

Unfortunately, the value of $\alpha$ is helpful only if it is very close to the actual maximum clique size. However, polynomial time algorithms can be used to find very good values for $\alpha$ (e.g. [1]).

**Example.** Consider the graph in Figure 1, with the steps of the algorithm shown in Table 1. We have found the maximum clique for this graph $\{3, 2, 4, 7\}$, with size 4.

Table 1

| | | |
|---|---|---|
| 1. Initialization orders the nodes 1 5 8 3 2 4 6 7 | | |
| 2. Depth 1: | **1** 5 8 3 2 4 6 7 | (the boldfaced node $v_{di}$ is expanded) |
| Depth 2: | **5** 2 4 | |
| Depth 3: | **4** | (cannot expand, so CBC is $\{1, 5, 4\}$, size 3) |
| Depth 2: | 5 **2** 4 | (since $d + (m - i) = 2 + (3 - 2) = 3 \leqslant 3$, we prune) |
| We are now finished with node 1 | | The largest clique containing this node is $\{1, 5, 4\}$ |
| 3. Depth 1: | 1 **5** 8 3 2 4 6 7 | |
| Depth 2: | **4** 6 | $(2 + (2 - 1) \leqslant$ CBC, so prune) |
| 4. Depth 1: | 1 5 **8** 3 2 4 6 7 | |
| Depth 2: | **3** 4 6 | |
| Depth 3: | **4** | (cannot expand, but size of $\{8, 3, 4\} \leqslant$ CBC) |
| Depth 2: | 3 **4** 6 | $(2 + (3 - 2) \leqslant$ CBC, so prune) |
| 5. Depth 1: | 1 5 8 **3** 2 4 6 7 | |
| Depth 2: | **2** 4 6 7 | |
| Depth 3: | **4** 7 | |
| Depth 4: | **7** | ($\{3, 2, 4, 7\}$ becomes our new CBC of size 4) |
| Depth 3: | 4 **7** | $(33 + (1 - 1) \leqslant 4$, so prune) |
| Depth 2: | 2 **4** 6 7 | $(2 + (4 - 2) \leqslant$ CBC, so prune) |
| 6. Depth 1: | 1 5 8 3 **2** 4 6 7 | $(1 + (8 - 5) \leqslant$ CBC and depth $= 1$, so stop) |

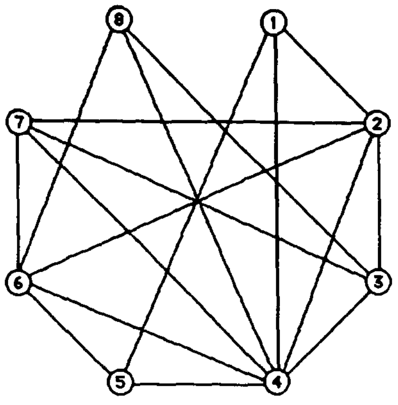

Fig. 1.

## 3. Computational result

In this section we present computational results to show the efficiency of our algorithm. The algorithm was implemented and run on an IBM 3090 computer without the use of its vectorization capabilities. The Fortran 77 program code, including the graph generating routines and the timing routines, is given in the Appendix.

Table 2

| Density | $n = 80$ | $n = 90$ | $n = 100$ |
|---|---|---|---|
| 0.50 | 0.05 | 0.09 | 0.14 |
| 0.60 | 0.14 | 0.31 | 0.52 |
| 0.70 | 0.65 | 1.50 | 2.82 |
| 0.80 | 2.84 | 9.85 | 32.25 |
| 0.90 | 62.91 | 316.40 | 481.10 |

Table 3

| Density | $n = 100$ | $n = 200$ | $n = 300$ | $n = 400$ | $n = 500$ |
|---|---|---|---|---|---|
| 0.10 | 0.01 | 0.03 | 0.08 | 0.19 | 0.39 |
| 0.20 | 0.01 | 0.08 | 0.30 | 0.76 | 1.68 |
| 0.30 | 0.02 | 0.26 | 1.23 | 4.04 | 11.21 |
| 0.40 | 0.06 | 0.93 | 6.12 | 23.16 | 75.45 |
| 0.50 | 0.14 | 4.16 | 46.04 | 235.68 | 1114.78 |

Table 2 contains results for graphs with $n = 80$, 90, 100 vertices and different densities. It is clear that the problem difficulty increases as the edge density increases.

For the problem with 100 vertices, and density of 0.90, using an initial incumbent of maxclique $- 1 = 31$, the CPU time required reduces to 341.62 secs. Similarly for the same problem, using incumbent = maxclique = 32, the problem was solved in 317.62 CPU secs (i.e., time to verify optimality).

Table 3 gives results for medium size problems ranging from graphs with 100 to 500 vertices and densities up to 50%.

Figure 2 shows the CPU time as a function of density and number of vertices.

Table 4

| Density | Maxcliq | B&B | Edges |
|---|---|---|---|
| 0.10 | 2 | 40 | 50000 |
| 0.20 | 22 | 373 | 100000 |
| 0.30 | 223 | – | 150000 |
| 0.40 | 3752 | – | 200076 |

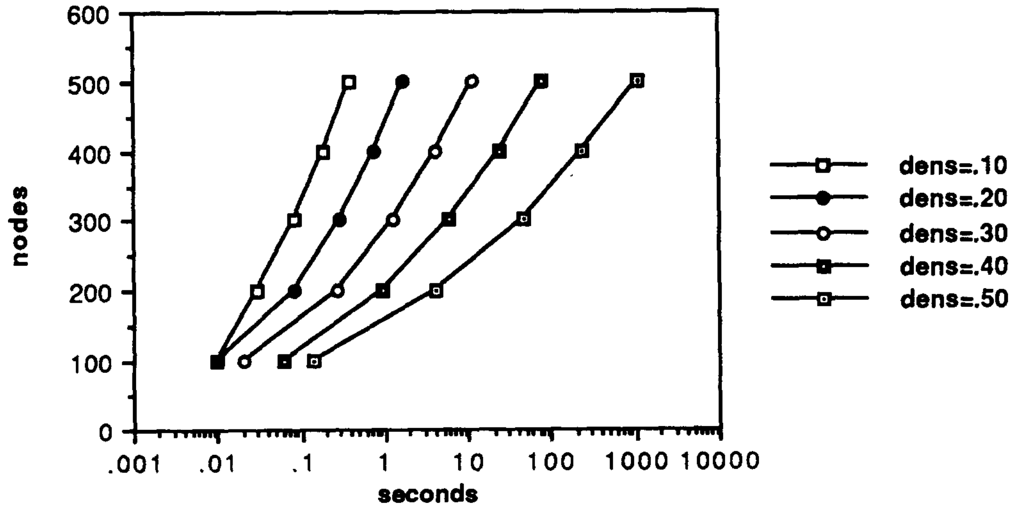

Fig. 2. CPU time for various densities.

Table 5
Computational results for Benchmarks 2000A, 2000B and 2000C, using VS Fortran on IBM 3090

| | A | B | C |
|---|---|---|---|
| Number of vertices | 2000 | 2000 | 2000 |
| Number of edges | 199452 | 400347 | 600318 |
| CPU time in seconds | 23 | 385 | 8443 |
| Maximum clique size | 6 | 8 | 11 |

Solution $A = (v_{16}, v_{148}, v_{397}, v_{539}, v_{789}, v_{1897})$
Solution $B = (v_1, v_{181}, v_{201}, v_{487}, v_{1021}, v_{1500}, v_{1538}, v_{1784})$
Solution $C = (v_{286}, v_{336}, v_{395}, v_{618}, v_{826}, v_{964}, v_{1429}, v_{1515}, v_{1558}, v_{1810}, v_{1852})$

Table 6
Computational results for Benchmarks 3000A, 3000B and 3000C, using VS Fortran on IBM 3090

| | A | B | C |
|---|---|---|---|
| Mumber of vertices | 3000 | 3000 | 3000 |
| Number of edges | 449162 | 899647 | 1035133 |
| CPU time in seconds | 98 | 307 | 6267 |
| Maximum clique size | 6 | 9 | 9 |

Solution $A = (v_1, v_{378}, v_{1188}, v_{1261}, v_{2003}, v_{2285})$
Solution $B = (v_{36}, v_{313}, v_{724}, v_{744}, v_{1075}, v_{2032}, v_{2243}, v_{2482}, v_{2959})$
Solution $C = (v_1, v_{29}, v_{569}, v_{601}, v_{1209}, v_{1309}, v_{2184}, v_{2459}, v_{2569})$

Table 4 gives the CPU time for three problems with 1000 vertices and 50000, 100000 and 150000 edges. The generation of these problems are described in [7] (the densities are 0.10, 0.20 and 0.30 respectively and the seed is 6551667.0). The heading Maxcliq gives the CPU time of our algorithm and B&B gives the CPU time obtained in [7]. Note that for the last problem vertex ordering was performed.

In Table 5 we consider three large-scale problems with 2000 vertices and densities 0.10, 0.20 and 0.30. The seeds for generating the random graphs are 3.1567, 6.1754 and 3.1856 respectively. Since these test problems are of low density, no vertex ordering was performed.

Next (Table 6) we consider three large-scale problems with 3000 vertices and densities 0.10, 0.20 and 0.23. The seeds are the same as above.

## 4. Concluding remarks

We have shown that a simple algorithm implemented very efficiently can solve very large maximum clique problems. Since this algorithm is very simple to implement, it can be used as a basis for computational comparison when different algorithms are used.

The proposed algorithm can be easily parallellized, since vertex $v_{1k}$ (depth 1) can be processed independently of vertices $v_{11}, \ldots, v_{1(k-1)}$.

The performance of the algorithm for dense problems can be improved by reapplication of vertex ordering at depths $> 1$. In addition, the algorithm will be faster when a good heuristic is used to determine $\alpha$.

## Acknowledgments

We would like to thank Gregory Rodgers for writing the timing subroutine and Wayne Carson for useful discussions on the implementation.

## References

[1] E. Balas and C.S. Yu, "Finding the maximum clique in an arbitrary graph", *SIAM J. Comput.* **15**/4, 1054–1068 (1986).

[2] P. Berman and G. Schnitger, *On the Complexity of Approximating the Independent Set Problem*, Springer-Verlag, Lecture Notes in Computer Science 349, 1989, 256–267.

[3] C. Bron and J. Kerbosch, "Finding all cliques of an undirected graph", *Comm. ACM* **16**/6, 575–577 (1973).

[4] M. Gendreau, J.-C. Picard and L. Zubieta, *An Efficient Implicit Enumeration Algorithm for the Maximum Clique Problem*, Springer-Verlag, Lecture Notes in Economics and

Mathematical Systems 304, A. Kurzhanski et al. (eds.), 1988, 79–91.

[5] L. Gerhards and W. Lindenberg, "Clique detection for nondirected graphs: Two new algorithms", *Computing* **21**, 295–322 (1979).

[6] E. Loukakis and C. Tsouros, "Determining the number of internal stability of a graph", *Internat. J. Comput. Math.* **11**, 232–248 (1982).

[7] P.M. Pardalos and G.P. Rodgers, "A branch and bound algorithm for the maximum clique problem", To appear in *Math. Programming* (1990).

[8] R.E. Tarjan and A.E. Trojanowski, "Finding a maximum independent set", *SIAM J. Comput.* **6**/3, 537–546 (1977).

[9] P. Turan, "On the theory of graphs", *Colloq. Math.* **3**, 19–30 (1954).

## Appendix

```
C      AN EFFICIENT PARTIAL ENUMERATIVE ALGORITHM FOR THE
C      MAXIMUM CLIQUE PROBLEM
C      RANDY CARRAGHAN AND PANOS PARDALOS
C      COMPUTER SCIENCE DEPARTMENT, PENN STATE UNIVERSITY
C      MARCH 15, 1989

       PARAMETER (MAXN=1000, MAXM=1000)
       INTEGER*2  N,ROW,COL
       LOGICAL*1  MAT(MAXN,MAXN),TEMPLOG
       INTEGER    EDGES
       REAL       DENS
       REAL*8     SEED
       INTEGER*2  ACTNODE(MAXN),TEMP
       INTEGER*2  ADJ(MAXM,MAXN)
       INTEGER*2  NODE,MIN,MINNODE,EDGE(MAXN)
       INTEGER*2  START(MAXN),LAST(MAXN),D,DTEMP,MAXCLIQUE,BEST(MAXN)
       REAL       T1,T2,SORTTIME,TRAVTIME
       REAL       TOTTIME

C      MAXN       MAXIMUM NUMBER OF NODES IN GRAPH
C      MAXM       GREATER OR EQUAL TO THE ACTUAL SIZE OF MAXCLIQUE
C      N          NUMBER OF NODES IN GRAPH
C      MAT        ADJACENCY MATRIX
C      ACTNODE    ACTUAL NODES IN ORIGINAL MATRIX
C      ADJ        REPRESENTS NODES FOUND AT ALL DEPTHS
C      MAXCLIQUE  SIZE OF LARGEST CURRENTLY KNOWN CLIQUE
C                 (INITIALLY WE MAY SET MAXCLIQUE=0)
C      BEST       NODES IN LARGEST CURRENTLY KNOWN CLIQUE
C      D          CURRENT DEPTH OF TRAVERSAL
C      START      NODE CURRENTLY BEING EXPANDED AT DEPTH D
C      LAST       LAST NODE TO (POSSIBLY) BE EXPANDED AT DEPTH D

C      -- GENERATION OF RANDOM GRAPH

       WRITE(*,*) 'ENTER N, DENSITY, INCUMBENT, SEED'
       READ(*,*) N,DENS,MAXCLIQUE,SEED
       EDGES = 0
       DO 10 ROW = 1,N-1
         DO 15 COL = ROW+1,N
           IF (GGUBFS(SEED).LT.DENS) THEN
             EDGES = EDGES + 1
             MAT(ROW,COL) = .TRUE.
             MAT(COL,ROW) = .TRUE.
           ELSE
             MAT(ROW,COL) = .FALSE.
             MAT(COL,ROW) = .FALSE.
           END IF
15       CONTINUE
10     CONTINUE
       DO 20 ROW = 1,N
         MAT(ROW,ROW) = .FALSE.
20     CONTINUE
       WRITE(*,*) ' '
       WRITE(*,85) 'TOTAL EDGES      ',EDGES

C      -- MAINTAIN POINTERS TO ORIGINAL MATRIX

       CALL SECOND(T1)
       DO 25 NODE = 1,N
         ACTNODE(NODE) = NODE
```

```
25    CONTINUE

C     -- ORDER NODES BY INCREASING EDGE DENSITY

      IF (DENS.GE..40) THEN
        DO 30 ROW = 1,N
          EDGE(ROW) = 0
          DO 35 COL = 1,N
            IF (MAT(ROW,COL)) EDGE(ROW) = EDGE(ROW)+1
35        CONTINUE
30      CONTINUE
        DO 40 NODE = 1,N-2
          MIN = N
          DO 45 ROW = NODE,N
            IF (EDGE(ROW).LT.MIN) THEN
              MIN = EDGE(ROW)
              MINNODE = ROW
            END IF
45        CONTINUE
          EDGE(MINNODE) = EDGE(NODE)
          IF (NODE.NE.MINNODE) THEN
            TEMP = ACTNODE(NODE)
            ACTNODE(NODE) = ACTNODE(MINNODE)
            ACTNODE(MINNODE) = TEMP
            DO 50 ROW = 1,N
              TEMPLOG = MAT(ROW,NODE)
              MAT(ROW,NODE) = MAT(ROW,MINNODE)
              MAT(ROW,MINNODE) = TEMPLOG
50          CONTINUE
            DO 55 COL = 1,N
              TEMPLOG = MAT(NODE,COL)
              MAT(NODE,COL) = MAT(MINNODE,COL)
              MAT(MINNODE,COL) = TEMPLOG
55          CONTINUE
          END IF
          DO 60 COL = NODE,N
            IF (MAT(NODE,COL)) EDGE(COL) = EDGE(COL) - 1
60        CONTINUE
40      CONTINUE
      END IF
      CALL SECOND(T2)
      SORTTIME = T2 - T1

      CALL SECOND(T1)
      D = 1
      START(1) = 0
      LAST(1) = N
      DO 65 COL = 1,N
        ADJ(1,COL) = COL
65    CONTINUE

C     -- MAIN ALGORITHM

70    START(D) = START(D) + 1
      IF ((D+LAST(D)-START(D)).GT.MAXCLIQUE) THEN
        DTEMP = D
        D = D + 1
        START(D) = 0
        LAST(D) = 0
C       -- DETERMINE NODE FOR NEXT DEPTH
        DO 75 COL = (START(DTEMP)+1),LAST(DTEMP)
          IF (MAT(ADJ(DTEMP,START(DTEMP)),ADJ(DTEMP,COL))) THEN
            LAST(D) = LAST(D) + 1
            ADJ(D,LAST(D)) = ADJ(DTEMP,COL)
            END IF
75        CONTINUE
C         -- IF THE NEXT DEPTH DOESN'T CONTAIN ANY NODES, SEE IF A NEW
C         -- MAXCLIQUE HAS BEEN FOUND AND RETURN TO PREVIOUS DEPTH
          IF (LAST(D).EQ.0.) THEN
            D = D - 1
            IF (D.GT.MAXCLIQUE) THEN
              MAXCLIQUE = D
              DO 80 COL = 1,D
                BEST(COL) = ADJ(COL,START(COL))
80            CONTINUE
            END IF
          END IF
        ELSE
C         -- PRUNE, FURTHER EXPANSION WOULD NOT FIND A BETTER INCUMBENT
          D = D - 1
        END IF
C       -- CONTINUE TRAVERSAL UNTIL A DEPTH OF ZERO IS REACHED
        IF (D.GT.0.) GOTO 70
        CALL SECOND(T2)
        TRAVTIME = T2 - T1

C       -- OUTPUT THE MAXIMUM CLIQUE FOUND IN GRAPH

        WRITE(*,85) 'MAXCLIQUE SIZE  ',MAXCLIQUE
        WRITE(*,*) ' '
        WRITE(*,*)  (ACTNODE(BEST(I)), I = 1,MAXCLIQUE)
        WRITE(*,*) ' '
        WRITE(*,90) 'ORDER MATRIX     ',SORTTIME
        WRITE(*,90) 'TRAVERSE TREE    ',TRAVTIME
        WRITE(*,*)  '                  ---------'
        TOTTIME = SORTTIME + TRAVTIME
        WRITE(*,90) 'TOTAL TIME       ',TOTTIME
85      FORMAT(1X,A,I10)
90      FORMAT(1X,A,F10.4)
        STOP
        END

C       -- IMSL RANDOM NUMBER GENERATOR

        REAL FUNCTION GGUBFS (DSEED)
        DOUBLE PRECISION   DSEED
        DOUBLE PRECISION   D2P31M,D2P31
        DATA               D2P31M/2147483647.D0/
        DATA               D2P31 /2147483648.D0/
        DSEED = DMOD(16807.D0*DSEED,D2P31M)
        GGUBFS = DSEED / D2P31
        RETURN
        END

C       -- DETERMINE CPU TIME FROM IBM 3090 CMS
C       -- (ROUTINE DEPENDENT UPON 'GLOBAL TXTLIB UTILITY' FROM CMS)

        SUBROUTINE SECOND(ECPU)
        LOGICAL*1 DATTIM(23)
        REAL*4 ECPU,ETIME,ETCPU
        INTEGER LD
        LD = 23
        CALL DATETM(DATTIM,LD,ECPU,ETIME,ETCPU)
        RETURN
        END
```