AN IMPLEMENTATION OF A PSEUDO-PERIPHERAL NODE FINDER[†]

by

Alan George*

and

Joseph W. H. Liu**

Research Report CS-76-44

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

November 1976

* Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

**Computer Services Division
City of Mississauga
Mississauga, Ontario, Canada

[†] Research supported in part by Canadian National Research Council grant A8111.

§1 Introduction

Many algorithms for finding orderings for symmetric matrices operate on the corresponding symmetric graph. These algorithms often require one or more "starting nodes", and for some algorithms experience suggests that nodes which are at maximum or nearly maximum distance apart are good candidates [4, 6, 7, 8, 9, 10]. In a recent paper Gibbs, Poole and Stockmeyer [8] provide a novel heuristic algorithm for finding such nodes. Our objective in this paper is to describe some modifications to their original algorithm and to present an implementation of the modified scheme. We also include some experiments illustrating the importance of the modifications.

We now give some formal definitions and a precise statement of the problem. Let $G = (X,E)$ be an undirected graph with the set X of nodes and the set E of unordered edges. A path of length k is an ordered set of distinct nodes $(x_0,x_1,\ldots,x_k)$ where $\{x_{i-1},x_i\} \in E$ for $1 \leq i \leq k$. A graph is connected if for each pair of distinct nodes, there is a path joining them. Throughout this paper, graphs are assumed to be connected unless we state otherwise.

Consider a connected graph G. The distance $d(x,y)$ between two nodes x and y in G is defined to be the length of a shortest path connecting them. Following Berge [2], we define the eccentricity of a node x to be the quantity

$$\ell(x) = \max \{d(x,y) \mid y \in X\}.$$

The diameter of G is then defined as

$$\delta(G) = \max \{\ell(x) \mid x \in X ,$$

or equivalently,

$$\delta(G) = \max \{d(x,y) \mid x,y \in X\}.$$

A node $x \in X$ is said to be a peripheral node if its eccentricity is equal to the diameter of the graph, that is, $\ell(x) = \delta(G)$. Following Gibbs et. al [8], we define a pseudo-peripheral node to be one whose eccentricity is close to $\delta(G)$.

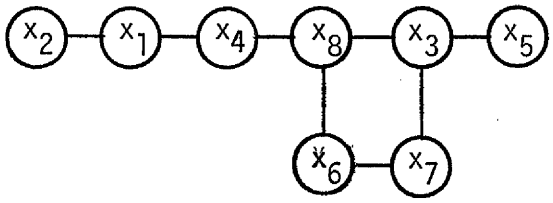


Figure 1.1 An 8-node graph.

Figure 1.1 is an example of a graph with 5 as its diameter. The nodes x_2, x_5 and x_7 are the only peripheral nodes.

With this terminology, our objective may be restated as an implementation of an efficient heuristic algorithm to determine pseudo-peripheral nodes.

It should be emphasized that both the original algorithm of Gibbs et. al [8] and our modified version are heuristic; neither is guaranteed to find a peripheral node, or even one that is close to being peripheral. Nevertheless, the nodes produced usually do have high eccentricity, and are effective as starting nodes for many ordering algorithms. In any case, except for some fairly trivial situations, there is no result which says that peripheral nodes are any better as starting nodes than the nodes found by our algorithm anyway. Finally, in many situations it is probably too expensive to find peripheral nodes even if it was known to be desirable to use them, since the best known algorithm for finding such nodes has a time complexity bound of $O(|X|^2)$ [11]. In some sparse matrix applications this would dominate all other phases of the computation [7].

§2 Some Related Definitions and Notations

In this section, we introduce some more definitions and notations that are useful in the description of the algorithm. Let $G = (X,E)$ be a connected graph and consider a subset $Y \subset X$. The adjacent set of Y is defined to be the set

$$Adj(Y) = \{x \in X\backslash Y \mid \{x,y\} \in E \text{ for some } y \in Y\}.$$

When $Y = \{y\}$, we use the notation $Adj(y)$ instead of $Adj(\{y\})$. The degree of a node y is then the number $|Adj(y)|$, where $|S|$ is the cardinality of the set S.

A graph $G' = (X',E')$ is said to be a subgraph of $G = (X,E)$ if $X' \subseteq X$ and $E' \subseteq E$. For a subset $Y \subset X$, the section graph of Y in G is the subgraph $G(Y) = (Y,E(Y))$, where

$$E(Y) = \{\{y_1,y_2\} \in E \mid y_1,y_2 \in Y\}.$$

Although the graph G is assumed to be connected, its section subgraphs may be disconnected. When a section graph $G(Y)$ is disconnected, it consists of two or more connected components.

A key construct in the algorithm described in section 3 is the rooted level structure [1]. Consider a node $x \in X$. The rooted level structure at x is the partitioning of the node set:

$$\mathcal{L}(x) = \{L_0(x), L_1(x),\ldots,L_{\ell(x)}(x)\},$$

where

$$L_0(x) = \{x\},$$

and

$$L_i(x) = Adj\left(\bigcup_{k=0}^{i-1} L_k(x)\right), \quad 1 \leq i \leq \ell(x).$$

The number $\ell(x)$, which is precisely the eccentricity of the node x, is sometimes referred to as the length of $\mathcal{L}(x)$. It can be verified readily that

$$\bigcup_{k=0}^{\ell(x)} L_k(x) = X.$$

The <u>width</u> of $\mathcal{L}(x)$ is defined to be

$$\max \{|L_i(x)| : 0 \leq i \leq \ell(x)\}.$$

For convenience, we shall denote this quantity by $w(x)$.

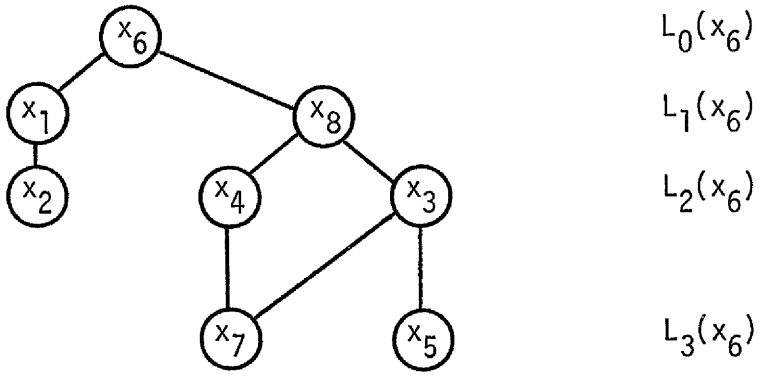


Figure 2.1 A rooted level structure $\mathcal{L}(x_6)$.

In figure 2.1, we have the rooted level structure of the graph of figure 1.1, rooted at the node x_6. Note that $\ell(x_6) = 3$ and $w(x_6) = 3$.

§3 Description of the algorithm and its modifications

§3.1 The Original Algorithm

In [8], Gibbs, Poole and Stockmeyer designed an algorithm to determine pseudo-peripheral nodes. Their algorithm is a finite iterative process based on the following observation:

$$y \in L_{\ell(x)}(x) \Rightarrow \ell(x) \leq \ell(y).$$

In our terminology, their algorithm can be described as follows:

Step 1 (Initialization): Choose a node r of minimum degree.

Step 2 (Generation of level structure): Construct the rooted level structure at r: $\mathcal{L}(r) = \{L_0(r), L_1(r),\ldots,L_{\ell(r)}(r)\}$.

Step 3 (Sort the last level): Sort the nodes in $L_{\ell(r)}(r)$ in increasing order of degree.

Step 4 (Test for termination): For $x \in L_{\ell(r)}(r)$ in increasing order of degree, generate $\mathcal{L}(x) = \{L_0(x), L_1(x),\ldots,L_{\ell(x)}(x)\}$. If $\ell(x) > \ell(r)$, set $r \leftarrow x$ and go to step 3.

Step 5 (Exit): The node r is the pseudo-peripheral node determined.

§3.2 Modification by Short-Circuiting

For a given graph $G = (X,E)$ and a node $x \in X$, the amount of work required to generate the rooted level structure $\mathcal{L}(x)$ is atleast $O(|E|)$, since each edge has to be inspected at least once. If r is the thus determined pseudo-peripheral node, it follows that the algorithm requires at least

$$|L_{\ell(r)}(r)| \cdot O(|E|)$$

units of work. How can then the algorithm be speeded up while producing reasonably good pseudo-peripheral nodes?

In their actual implementation of their algorithm [3], Gibbs et. al. improved the efficiency by introducing a "short circuit" technique, where wide level structures are rejected as soon as they are detected. This idea is legitimate in view of the observation that long level structures are usually narrow. (In any case, in their application their primary interest was in finding narrow level structures, rather than ones having a large length.)

Their modification can be described by changing step 4 as follows:

Step 4' (Test for termination): For $x \in L_{\ell(r)}(r)$ in increasing order of degree, generate $\mathcal{L}(x) = \{L_0(x), L_1(x),\ldots,L_{\ell(x)}(x)\}$. If $w(x) \leq w(r)$ and $\ell(x) > \ell(r)$, set $r \leftarrow x$ and go to step 2.

Note that step 4' can be implemented in such a way so that if the condition $w(x) \leq w(r)$ is violated, only part of the rooted level structure $\mathcal{L}(x)$ needs to be generated. This can be done by testing if $|L_i(x)| > w(r)$ as each level $L_i(x)$ is generated. This modification has the effect of reducing the second factor in the expression $|L_{\ell(r)}(r)| \cdot O(|E|)$.

§3.3 Modification by Shrinking

In this subsection, we introduce another modification, the objective of which is to reduce the number of level structures generated. The modified algorithm is described in detail as follows.

Step 1 (Initialization): Choose an arbitrary node r in X.

Step 2 (Generation of level structure): Construct the rooted level structure at r: $\mathcal{L}(r) = \{L_0(r), L_1(r),\ldots,L_{\ell(r)}(r)\}$.

Step 3 (Shrinking): Find all the connected components in the section graph $G(L_{\ell(r)}(r))$.

Step 4 (Test for termination): For each connected component C in $G(L_{\ell(r)}(r))$, find a node x of minimum degree and generate its rooted level structure $\mathcal{L}(x)$.

If $\ell(x) > \ell(r)$, put $r \leftarrow x$ and go to step 3.

Step 5 (Exit): The node r is the pseudo-peripheral node determined.

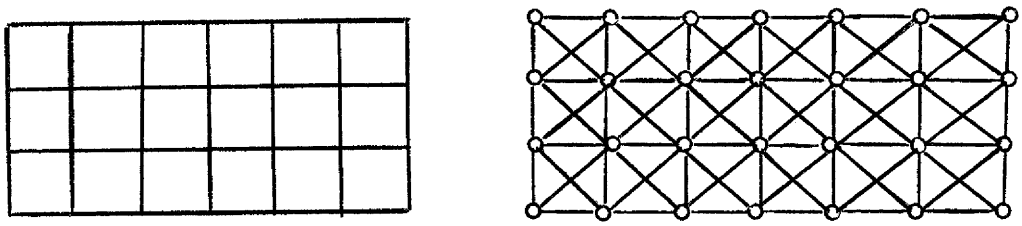
Here, instead of considering each and every node in $L_{\ell(r)}(r)$, we only pick representatives from $L_{\ell(r)}(r)$. This modification improves the execution time substantially and our experience is that it does not affect the outcome much. In the next section, we include some experimental results that illustrate these points.

In addition, note that our step one chooses an arbitrary node, rather than one of minimum degree. Thus, we avoid computing the degrees of the nodes, and finding one of minimum degree. We found that this modification usually reduced total execution time, and made little or no difference to the results.

§4 Some Experiments

In this section, we present some numerical experiments comparing the performances of the algorithm and its modifications as discussed in section 3. The algorithms have been implemented and tested on a collection of graph problems that arise in practical applications. To see how the modified schemes compare to the original algorithm, we consider an example in detail. The result is typical in all the examples that we tried and it reflects to some extent the general nature of the algorithms.

Consider an n by 2n regular grid. Associated with it is a graph with $N = (n+1)(2n+1)$ nodes located at the intersections of the grid lines. Nodes are considered to be connected if and only if they share a common small "square". Figure 4.1 contains an example of a 3 x 6 grid and its associated graph.



Grid Graph

Figure 4.1 An 3 x 6 grid and its associated graph.

To obtain an asymptotic behaviour of the algorithms on this grid problem, we applied them to graphs with different subdivision factor n. The numerical results are reported in table 4.1. Here, the execution time is in seconds on an IBM 360/75 computer. The programs were run under the University of Waterloo WATFIV debugging compiler. A Fortran listing of the subroutines used appears in the Appendix.

n	N	Original Algorithm		By Short Circuiting		By Shrinking	
		Time	Time/(n+2)N	Time	Time/($\frac{n}{2}$+2)N	Time	Time/3N
5	66	0.47	1.02(-3)	0.34	1.14(-3)	0.22	1.11(-3)
10	231	2.96	1.07(-3)	1.85	1.14(-3)	0.71	1.02(-3)
15	496	8.56	1.02(-3)	5.24	1.11(-3)	1.47	0.98(-3)
20	861	19.36	1.02(-3)	11.50	1.11(-3)	2.58	0.99(-3)

Table 4.1 Tabulated results for the n by 2n regular grid graph.

The execution time of the modified scheme by short-circuiting nearly halves that of the original algorithm. This is to be expected since most of the rooted level structures in the short-circuiting scheme are only half-formed.

However, both schemes have a running time proportional to $N^{3/2}$, where N is the number of nodes in the graph. The modification by shrinking is dramatically faster, and it has a time complexity bound of O(N).

The differences in the execution time between the algorithms would not be meaningful unless they produce comparable results. In fact, although the pseudo-peripheral nodes determined by the three schemes are different, they all have the same eccentricity, namely 2n. Indeed, the method by shrinking compares very favorably with the other two.

Note that if we had chosen an n by (n+1) grid, the short-circuit technique would save almost nothing, and the advantage of our scheme would have been even more dramatic.

Results of the same nature were obtained for other test examples, which included the set of sparse graphs collected by E.H. Cuthill and G.C. Everstine, used in [8]. In all the test examples, nodes generated by the schemes have <u>equal</u> eccentricity and the method by shrinking require substantially less execution time than the original algorithm.

§5 Concluding Remarks

The need to find nodes with large eccentricity arises in many heuristic ordering algorithms for sparse matrices. We have made a suggestion to modify the heuristic algorithm by Gibbs et. al for determining such nodes. Experiments have shown that the modified algorithm requires significantly less time for execution and produces nodes of comparable eccentricity. In many practical cases of interest, it reduces the execution time from $O(N^{3/2})$ to $O(N)$.

§6 References

[1] I. Arany, W.F. Smyth and L. Szoda, "An improved method for reducing the bandwidth of sparse symmetric matrices", in Information Processing 71: Proceedings of IFIP Congress, North-Holland, Amsterdam, 1972.

[2] C. Berge, The Theory of Graphs and its Applications, John Wiley & Sons Inc., New York, 1962.

[3] H.L. Crane, Jr., N.E. Gibbs, W.G. Poole, Jr., and P.K. Stockmeyer, "Matrix bandwidth and profile minimization", Report No.75 - 9, ICASE Report (1975).

[4] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices", Proc. 24th Nat. Conf., Assoc. Comput. Mach.,ACM Publ. P-69, 1122 Ave. of the Americas, New York, N.Y. (1969).

[5] G.C. Everstine, "The BANDIT computer program for the reduction of matrix bandwidth for NASTRAN', NSRDC Report 3827 (1972).

[6] Alan George and Joseph W.H. Liu, "Algorithms for matrix partitioning and the numerical solution of finite element systems", Rept. CS-76-30, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, (1976).

[7] Alan George and Joseph W.H. Liu, "An automatic nested dissection algorithm for irregular finite element problems", Rept. CS-76-38, Department of Computer Science, University of Waterloo, Waterloo Ontario, Canada, (1976).

[8] N.E. Gibbs, W.G. Poole and P.K. Stockmeyer, "An algorithm for reducing the bandwidth and profile of a sparse matrix", SIAM J. Numer. Anal., 13 (1976), pp.236-250.

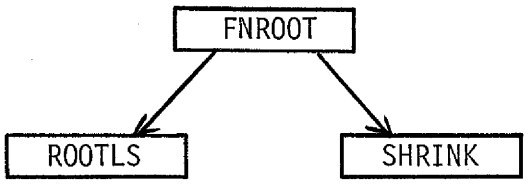
[9] Joseph W.H. Liu, "On reducing the profile of sparse symmetric matrices", Rept. CS-76-07, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, (1976).

[10] Joseph W.H. Liu and Andrew H. Sherman, "Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices", SIAM J. Numer. Anal., 13 (1975),pp.198-213.

[11] W.F. Smyth and W.M.L. Benzi, "An algorithm for finding the diameter of a graph", IFIP Congress 74, Vol.3, North-Holland Publ. Co., 1974, pp.500-503.

Appendex

We include here a FORTRAN implementation of our modified algorithm. It consists of three subroutines whose control relationship is as depicted below.



The subroutine FNROOT is the driver routine that returns a pseudo-peripheral node in a masked subgraph. It calls ROOTLS and SHRINK to generate rooted level structures and to shrink last level node sets respectively.

In the implementation, the graph is stored as a sequence of adjacency lists. The lists are kept consecutively in a single array "ADJNCY". An index array "XADJ" is used to mark the beginnings of the adjacency list. Thus, the nodes adjacent to the i-th node are given by {ADJNCY(k) | k = XADJ(i), XADJ(i)+1,...,XADJ(i+1)-1}. The rooted level structure is stored in the same manner by an array pair (LS, XLS).

```
C*************************************************************
C
C     SUBROUTINE FNROOT DETERMINES A PSEUDO PERIPHERAL NODE
C     FOR THE 'MASK'-ED COMPONENT CONTAINING THE NODE 'ROOT'.
C     A MODIFIED VERSION OF THE SCHEME BY GIBBS, POOLE, AND
C     STOCKMEYER (SIAM J. NUMER. ANAL., 13 (1976), PP 236-250.)
C     IS USED.
C
C     THE GRAPH IS STORED AS A SEQUENCE OF ADJACENCY LISTS,
C     STORED CONSECUTIVELY IN THE ARRAY ADJNCY.  THE POSITION
C     OF THE ADJACENCY LIST FOR THE I-TH NODE IS GIVEN BY
C     XADJ(I). THUS, THE NODES ADJACENT TO NODE I ARE GIVEN BY
C     ADJNCY(K), K = XADJ(I), XADJ(I)+1, ... XADJ(I+1)-1.
C
C     THE ARRAY MASK IS USED TO MASK OFF A SUBGRAPH OF THE
C     ONE STORED IN THE ARRAY PAIR (ADJNCY, XADJ). 'FNROOT'
C     IGNORES NODES FOR WHICH MASK(I)=0.
C
C     THE ARRAY PAIR (LS, XLS) IS USED TO STORE LEVEL STRUCTURES.
C     NODES IN LEVEL I ARE FOUND IN LS(K) FOR K EQUAL TO
C     XLS(I), XLS(I)+1, ..., XLS(I+1)-1.
C
C     THE ARRAY LASTLV IS TEMPORARY STORAGE USED TO STORE
C     THE NODES OF THE LAST LEVEL OF A LEVEL STRUCTURE.
C
C     THE NUMBER NLVL IS THE NUMBER OF LEVELS IN THE LEVEL
C     STRUCTURE OF MAXIMUM LENGTH SO FAR FOUND.
C
C     ON EXIT, ROOT CONTAINS THE PSEUDO-PERIPHERAL NODE
C     FOUND, AND THE LEVEL STRUCTURE ROOTED AT THAT NODE
C     IS STORED IN THE ARRAY PAIR (LS, XLS). NLVL IS THE
C     LENGTH OF THAT LEVEL STRUCTURE.
C
C*************************************************************
C
      SUBROUTINE FNROOT(ROOT, ADJNCY, XADJ, MASK,
     *                  LS, XLS, NLVL, LASTLV)
C
C-----------------------------------------------------------------
      INTEGER ADJNCY(1), XADJ(1), MASK(1), LS(1), XLS(1),
     1        LASTLV(1), I, J, ROOT, NLVL, NUNLVL, LLSIZE
C-----------------------------------------------------------------
C
C     DETERMINE ROOTED LEVEL STRUCTURE AT 'ROOT'.
C
      CALL ROOTLS(ROOT, ADJNCY, XADJ, MASK, LS, XLS, NLVL)
C
      IF (NLVL .EQ. 1) RETURN
C
C     COPY THE LAST LEVEL OF LS TO VECTOR 'LASTLV'.
C
  100 J = XLS(NLVL)
      LLSIZE = XLS(NLVL+1) - J
C
      DO 200 I = 1, LLSIZE
         LASTLV(I) = LS(J)
         J = J + 1
  200 CONTINUE
C
C     LET G' BE THE SECTION GRAPH PRESCRIBED BY THE NODES
C     OF THE LAST LEVEL (LASTLV). CALL 'SHRINK' TO SHRINK
C     EACH CONNECTED COMPONENT OF G' TO A NODE OF MINIMUM
C     DEGREE IN EACH COMPONENT.
C
      IF ( LLSIZE .GT. 1 )
     1    CALL SHRINK(ADJNCY, XADJ, MASK, LASTLV, LLSIZE, LS)
C
C     FOR EACH NODE IN THE (SHRUNK) LAST LEVEL, GENERATE ITS
C     ROOTED LEVEL STRUCTURE.  COMPARE ITS LENGTH
C     WITH THE PREVIOUS LEVEL STRUCTURE. IF A LONGER ONE IS
C     FOUND, USE ITS LAST LEVEL TO START OVER (GO TO 100).
C
      DO 300 I = 1, LLSIZE
         ROOT = LASTLV(I)
C
         CALL ROOTLS(ROOT, ADJNCY, XADJ, MASK, LS, XLS, NUNLVL)
C
         IF (NUNLVL .EQ. NLVL) GO TO 300
C
         NLVL = NUNLVL
         GO TO 100
C
  300 CONTINUE
C
      RETURN
      END
```

```
C*****************************************************************
C
C     ROOTLS GENERATES THE ROOTED LEVEL STRUCTURE AT THE NODE
C     'ROOT'.
C
C     THE GRAPH IS STORED AS A SEQUENCE OF ADJACENCY LISTS,
C     STORED CONSECUTIVELY IN THE ARRAY ADJNCY.  THE POSITION
C     OF THE ADJACENCY LIST FOR THE I-TH NODE IS GIVEN BY
C     XADJ(I). THUS, THE NODES ADJACENT TO NODE I ARE GIVEN BY
C     ADJNCY(K), K = XADJ(I), XADJ(I)+1, ... XADJ(I+1)-1.
C
C     THE ARRAY MASK IS USED TO MASK OFF A SUBGRAPH OF THE
C     ONE STORED IN THE ARRAY PAIR (ADJNCY, XADJ). 'ROOTLS'
C     IGNORES NODES FOR WHICH MASK(I)=0.
C
C     THE ARRAY PAIR (LS, XLS) IS USED TO STORE THE LEVEL
C     STRUCTURE. NODES IN LEVEL I ARE FOUND IN LS(K),
C     FOR K = XLS(I), XLS(I)+1, ..., XLS(I+1)-1.
C
C     THE NUMBER NLVL IS THE NUMBER OF LEVELS IN THE LEVEL
C     STRUCTURE.
C
C*****************************************************************
C
      SUBROUTINE ROOTLS(ROOT, ADJNCY, XADJ, MASK, LS, XLS, NLVL)
C
C-----------------------------------------------------------------
      INTEGER ADJNCY(1), XADJ(1), MASK(1), LS(1), XLS(1),
     1        JSTRT, JSTOP, J, NBR, LVSIZE, NODE, ROOT,
     2        NLVL, I, LNBR, LBEGIN, LVLEND
C-----------------------------------------------------------------
C
C        INITIALIZATION ....
C
         MASK(ROOT) = 0
         LS(1) = ROOT
         NLVL = 0
         LVLEND = 0
         LNBR = 1
C
C        'LBEGIN' IS THE POINTER TO THE BEGINNING OF PRESENT
C        LEVEL; AND 'LVLEND' POINTS TO THE END OF THIS LEVEL.
C
  200    LBEGIN = LVLEND + 1
         LVLEND = LNBR
         NLVL = NLVL + 1
         XLS(NLVL) = LBEGIN
C
C        GENERATE THE NEXT LEVEL BY FINDING ALL THE MASKED
C        NEIGHBORS OF NODES IN PRESENT LEVEL.
C
         DO 400 I = LBEGIN, LVLEND
            NODE = LS(I)
            JSTRT = XADJ(NODE)
            JSTOP = XADJ(NODE + 1) - 1
C
            IF ( JSTOP .LT. JSTRT ) GO TO 400
C
            DO 300 J = JSTRT, JSTOP
               NBR = ADJNCY(J)
C
               IF (MASK(NBR) .EQ. 0) GO TO 300
C
               LNBR = LNBR + 1
               LS(LNBR) = NBR
               MASK(NBR) = 0
C
  300       CONTINUE
  400    CONTINUE
C
C        COMPUTE THE CURRENT LEVEL WIDTH.
C        IF IT IS NONZERO, GENERATE NEXT LEVEL.
C
         LVSIZE = LNBR - LVLEND
         IF (LVSIZE .GT. 0 ) GO TO 200
C
         XLS(NLVL+1) = LVLEND + 1
C
C        RESET MASK TO ONE FOR THE NODES IN THE LEVEL STRUCTURE.
C
         DO 500 I = 1, LNBR
            MASK(LS(I)) = 1
  500    CONTINUE
C
         RETURN
       END
```

```
C*************************************************************
C
C     THIS SUBROUTINE IS USED BY FNROOT TO CONDENSE INPUT
C     NODE SETS BY CHOOSING A NODE OF MINIMUM DEGREE FROM
C     EACH CONNECTED COMPONENT IN THE SECTION GRAPH SPECIFIED
C     BY THE NODES OF THE INPUT SET.  ON INPUT, THE LLSIZE
C     NODES OF THE INPUT SET ARE STORED IN LASTLV. ON OUTPUT,
C     THE NODES OF MINIMUM DEGREE, ONE FROM EACH COMPONENT,
C     ARE STORED IN LASTLV, AND LLSIZE IS SET TO THE NUMBER
C     OF SUCH NODES FOUND (NCOMP).
C
C     THE GRAPH IS STORED AS A SEQUENCE OF ADJACENCY LISTS,
C     STORED CONSECUTIVELY IN THE ARRAY ADJNCY.  THE POSITION
C     OF THE ADJACENCY LIST FOR THE I-TH NODE IS GIVEN BY
C     XADJ(I). THUS, THE NODES ADJACENT TO NODE I ARE GIVEN BY
C     ADJNCY(K), K = XADJ(I), XADJ(I)+1, ... XADJ(I+1)-1.
C
C     THE ARRAY MASK IS USED TO MASK OFF A SUBGRAPH OF THE
C     ONE STORED IN THE ARRAY PAIR (ADJNCY, XADJ). 'SHRINK'
C     IGNORES NODES FOR WHICH MASK(I)=0.
C
C     THE ARRAY XADJ IS USED AS A TEMPORARY MASK TO INDICATE
C     WHICH NODES IN THE MASK-ED SUBGRAPH ARE ALSO IN THE
C     SECTION GRAPH SPECIFIED BY THE NODES OF THE INPUT SET.
C
C*************************************************************
C
      SUBROUTINE SHRINK(ADJNCY, XADJ, MASK,
     1                  LASTLV, LLSIZE, TEMPCC)
C
C-------------------------------------------------------------
      INTEGER  ADJNCY(1), XADJ(1), MASK(1), LASTLV(1), TEMPCC(1),
     1         CCPTR, DEG, LLSIZE, NCOMP, INODE, NUM, MINPTR,
     2         CCSIZE, NODE, NBR, I, J, MINDEG, JSTRT, JSTOP
C-------------------------------------------------------------
C
C     MASK THE NODES IN THE LAST LEVEL
C
      DO 100 I = 1, LLSIZE
         J = LASTLV(I)
         XADJ(J) = -XADJ(J)
  100 CONTINUE
C
C     NCOMP COUNTS THE NUMBER OF COMPONENTS FOUND. THE
C     FINAL VALUE OF NCOMP IS THE RETURNED VALUE OF LLSIZE.
      NCOMP = 0
C
C     NUM COUNTS THE NODES AS THEY ARE FOUND, WHEN
C     NUM = LLSIZE WE CAN STOP.
      NUM = 0
C
      DO 500 I = 1, LLSIZE
         INODE = LASTLV(I)
C
C        CHECK TO SEE IF THE COMPONENT CONTAINING INODE
C        HAS ALREADY BEEN PROCESSED.
         IF ( XADJ(INODE) .GT. 0 ) GO TO 500
C
C        FIND THE COMPONENT IN LASTLV CONTAINING INODE.
C
         CCPTR = 1
         CCSIZE = 1
         TEMPCC(1) = INODE
         MINPTR = 1
         MINDEG = IABS(XADJ(INODE+1)) + XADJ(INODE)
         XADJ(INODE) = -XADJ(INODE)
C
  200    NODE = TEMPCC(CCPTR)
         DEG = 0
         JSTRT = XADJ(NODE)
         JSTOP = IABS(XADJ(NODE + 1)) - 1
C
         DO 300 J = JSTRT, JSTOP
            NBR = ADJNCY(J)
C
            IF ( MASK(NBR) .NE. 0 ) DEG = DEG + 1
C
            IF ( XADJ(NBR) .GT. 0 ) GO TO 300
C
            CCSIZE = CCSIZE + 1
            TEMPCC(CCSIZE) = NBR
            XADJ(NBR) = -XADJ(NBR)
C
  300    CONTINUE
C
C        CHECK IF THE DEGREE OF THIS NODE IS LESS THAN
C        THE LOWEST DEGREE THUS FAR IN THIS COMPONENT.
         IF ( DEG .GE. MINDEG ) GO TO 400
C
         MINDEG = DEG
         MINPTR = CCPTR
C
C        CHECK IF ALL NODES IN COMPONENT HAVE BEEN FOUND.
C
  400    CCPTR = CCPTR + 1
         IF (CCPTR .LE. CCSIZE) GO TO 200
C
C        ANOTHER COMPONENT HAS BEEN FOUND
C
         NCOMP = NCOMP + 1
         LASTLV(NCOMP) = TEMPCC(MINPTR)
         NUM = NUM + CCSIZE
         IF ( NUM .GE. LLSIZE ) GO TO 600
C
  500 CONTINUE
C
C     SET LLSIZE TO THE NUMBER OF COMPONENTS FOUND.
C
  600 LLSIZE = NCOMP
      RETURN
      END
```