INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 19, 1153-1181 (1983)

AUTOMATIC ELEMENT REORDERING FOR FINITE ELEMENT ANALYSIS WITH FRONTAL SOLUTION SCHEMES

S. W. SLOAN[†]
Department of Engineering Science, Parks Road, Oxford OX1 3PJ, England
AND
M. F. RANDOLPH[‡]
University Engineering Department, Trumpington Street, Cambridge CB2 1PZ, England

SUMMARY

This paper describes an element reordering algorithm which is suitable for use with a frontal solution package. The procedure is shown to generate efficient element numberings for a wide variety of test examples. In an effort to obtain an optimum elimination order, the algorithm first renumbers the nodes, and then uses this result to resequence the elements. This intermediate step is necessary because of the nature of the frontal solution procedure, which assembles variables on an element-by-element basis but eliminates them node by node. To renumber the nodes, a modified version of the King[1] algorithm is used. In order to minimize the number of nodal numbering schemes that need to be considered, the starting nodes are selected automatically by using some concepts from graph theory. Once the optimum numbering sequence has been ascertained, the elements are then reordered in an ascending sequence of their lowest-numbered nodes. This ensures that the new elimination order is preserved as closely as possible.

For meshes that are composed of a single type of high-order element, it is only necessary to consider the vertex nodes in the renumbering process. This follows from the fact that mesh numberings which are optimal for low-order elements are also optimal for high-order elements. Significant economies in the reordering strategy may thus be achieved. A computer implementation of the algorithm, written in FORTRAN IV, is given.

INTRODUCTION

In recent years, the frontal solution procedure has become increasingly popular as a means of solving the sparse symmetric matrix equations which often arise in finite element computations. A detailed discussion of the history and merits of the frontal approach has been given in a recent text by Irons and Ahmad,[2] and will not be repeated here. It suffices to note that it is generally as efficient as the more traditional type of bandwidth solver, both in terms of arithmetic and storage requirements, and is particularly suited to high-order elements which have midside or interior nodes.

When employed in a finite element context, the efficiency of a frontal scheme is dependent upon the ordering of the elements, with the ordering of the nodes being immaterial. This is because the equations are assembled and factorized on an element-by-element basis. In contrast, the efficiency of bandwidth-oriented schemes is solely a function of the nodal ordering. In the simplest type of bandwidth algorithm, where all matrix entries inside the bandwidth are stored and operated on, the nodes are labelled in an attempt to procure a small bandwidth. Indeed, this is essential for an economical solution.

† Research Fellow.
‡ Assistant Lecturer.

0029-5981/83/081153-29$02.90
© 1983 by John Wiley & Sons, Ltd.

Received 1 November 1981
Revised 2 June 1982

For problems which have few elements, or an uncomplicated topology, it is relatively straightforward to label elements so that the frontal procedure will perform efficiently. In more complex examples, however, such as large water distribution networks and two- and three-dimensional finite element grids, this is frequently difficult. The need to number finite element meshes efficiently is particularly important in nonlinear computations, where it is often necessary to solve a system of linear algebraic equations a large number of times. Classes of problems for which efficient solutions are necessary include implicit dynamic calculations and large-scale plasticity analysis. In a more general sense, the increasing use of microcomputers, which have limited fast store and relatively slow central processing units, has provided additional impetus for deriving efficient solution schemes.

THE ELIMINATION PROCESS

From an inspection of the Gaussian elimination procedure, it is possible to estimate the number of operations that are needed to solve a set of equations when employing the frontal method. This provides valuable insight in regard to the development of a heuristic algorithm for reordering the element numbers, such as that given by King.[1]

In most applications, the finite element method yields a sparse set of linear algebraic equations of the form

$$[K]\{x\}=\{f\} \tag{1}$$

where $[K]$ is a symmetric positive definite matrix of dimension $n\times n$, and $\{x\}$ and $\{f\}$ are vectors of dimension $n\times 1$. In the frontal solution method, the unknowns $\{x\}$ are determined by applying straightforward Gaussian elimination. Prior to the elimination phase, however, the overall coefficient matrix is not assembled explicitly. Instead, the assembly and elimination processes are interleaved, with each row and the associated column in $[K]$ being eliminated at the earliest opportunity. Simple examples which illustrate the frontal principle may be found in Hinton and Owen[3] and Irons and Ahmad.[2]

In equation (1) for any two rows s and i, we may write

$$\begin{aligned} \ldots K_{ss}x_s+\ldots K_{sj}x_j+\ldots &= f_s \\ \ldots K_{is}x_s+\,.\,.\,l\,K_{ij}x_j+\ldots &= f_i \end{aligned} \tag{2}$$

Using Gauss's process, the coefficient K_{is} may be eliminated by subtracting K_{is}/K_{ss} times row s row from i, i.e.

$$\begin{aligned} \ldots K_{ss}x_s+ \qquad\qquad \ldots K_{sj}x_j+\ldots &= f_s \\ \ldots 0+\ldots\left[K_{ij}-\frac{K_{is}}{K_{ss}}K_{sj}\right]x_j+\ldots &= f_i-\left[\frac{K_{is}}{K_{ss}}\right]f_s \end{aligned} \tag{3}$$

Note that row s is unaltered during the elimination of x_s from the remaining equations, and that column s of $[K]$ may be eliminated as soon as it is fully assembled. More generally, the Gaussian elimination procedure may be summarized by the equations

$$\begin{aligned} K_{ij}{}^{*} &= K_{ij}-\left[\frac{K_{is}}{K_{ss}}\right]K_{sj} \\ f_i{}^{*} &= f_i-\left[\frac{K_{is}}{K_{ss}}\right]f_s \end{aligned} \tag{4}$$

where the entry in the sth column of row i is being eliminated.

If, for the moment, symmetry is ignored, it is a relatively simple task to determine the number of operations† that are required to eliminate a variable from an $n \times n$ set of equations. From an inspection of the left-hand side of equation (3) it is apparent that we need one division to form the multiplier K_{is}/K_{ss}, and $n-1$ multiplications to compute all the 'starred' coefficients for the ith row (noting that no multiplication is necessary to compute K_{is}^* since it is identically zero). Hence the total number of operations that are required to eliminate x_s from all the equations is

$$n(n-1) \tag{5}$$

More generally, if the effect of the right-hand side is included, the number of operations is

$$(n-1)(n+1) \tag{6}$$

If the governing linear equations are symmetric, only the upper triangle of the coefficient matrix needs to be stored and operated on. This leads to significant savings in the total number of operations that are required to eliminate a variable from the left-hand side. Referring to equations (3) and (4), the total number of multiplications that are required to eliminate x_s from all the equations is

$$\tfrac{1}{2}(n)(n+1)-n$$

where the $-n$ term arises because none of the entries in row s needs to be modified. The number of divisions that are required to compute the multiplier for each row i, $i \neq s$, is equal to $n-1$. Hence, the total number of operations that are required to eliminate x_s from the left-hand side of the equations is

$$\tfrac{1}{2}(n)(n+1)-1 \tag{7}$$

If the effect of the right-hand side is included, the number of operations is increased to

$$\tfrac{1}{2}(n^2+3n-4) \tag{8}$$

For a symmetric matrix of coefficients $[K]$, equation (7) may be used to estimate the number of operations that are required to decompose the left-hand side when using a frontal solution procedure. If, at some stage during the elimination process, the number of active variables in the front is w, then the number of operations that are necessary to eliminate a variable from the front is simply

$$\tfrac{1}{2}(w)(w+1)-1 \tag{9}$$

If w_i denotes the value of w just prior to the elimination of variable x_i, then the total number of operations that are required to decompose the $[K]$ matrix is

$$T=\tfrac{1}{2}\sum_{i=1}^{n}(w_i^2+w_i-2) \tag{10}$$

In deriving equation (10), it has been assumed that no zeros occur inside the front during the elimination process. This assumption is a simplification, as the elimination of fully assembled equations may lead to the creation of zero rows/columns inside the upper triangle of the active frontal matrix. This is because, for the simplest form of frontal solver, the elimination of active variables may occur in any order (Figure 1). Although it is usual to try and fill these vacant locations with coefficients when assembling each new element, unfilled rows/columns may still persist inside the front. The occurrence of this phenomenon requires that a distinction

† Following the usual convention, an operation is defined as one multiplication or one division.

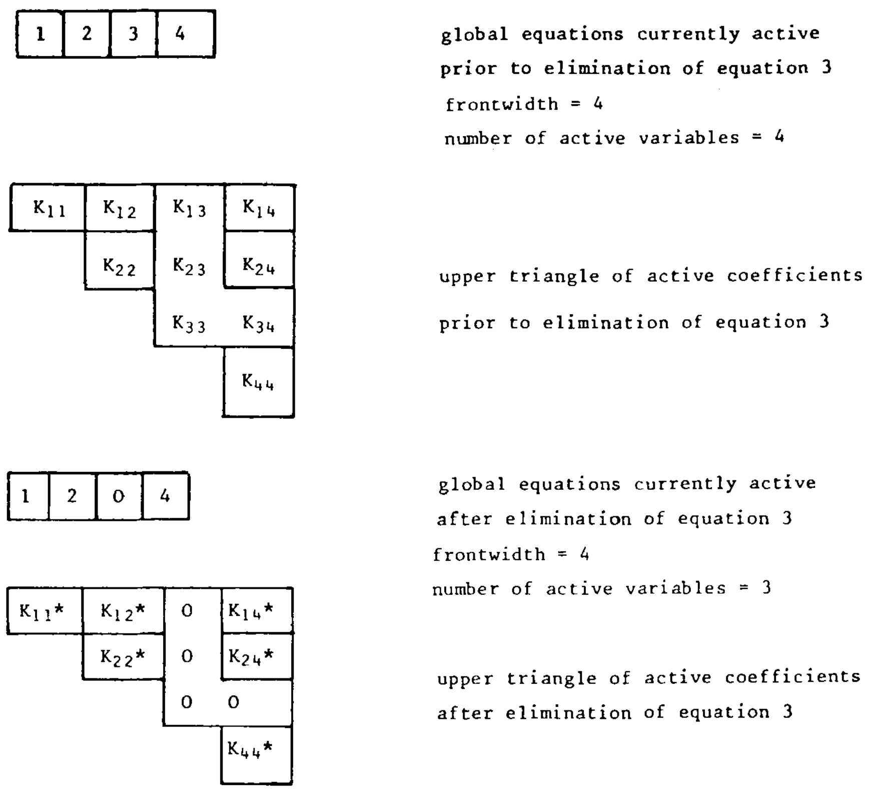

global equations currently active
prior to elimination of equation 3
frontwidth = 4
number of active variables = 4
upper triangle of active coefficients
prior to elimination of equation 3
global equations currently active
after elimination of equation 3
frontwidth = 4
number of active variables = 3
upper triangle of active coefficients
after elimination of equation 3

Figure 1. Introduction of zero rows/columns in upper triangle of active coefficients

be made between the number of active variables in the front and the actual current frontwidth. The actual current frontwidth is always greater than, or equal to, the number of active variables.

If, during the elimination process, unfilled zero rows/columns arise inside the upper triangle of coefficients, the number of operations given by equation (10) will be a *lower bound*. However, approximately half of the redundant operations introduced may be removed by checking for zero columns in the outer loop of the elimination routine. Since the current frontwidth is equal to the number of active variables when the latter is a maximum, an *upper bound* on the number of operations is given by

$$T = \frac{n}{2}(W^2 + W - 2) \tag{11}$$

where W is the maximum frontwidth.

A REVIEW OF ELEMENT RESEQUENCING STRATEGIES

Although a wealth of literature exists on heuristic algorithms for minimizing the bandwidth of sparse symmetric matrices, very little attention has been focused on the development of schemes aimed at minimizing the frontwidth. Broadly speaking, there are two different methods of approaching this problem. Each of these will be considered in turn.

The direct methods

The earliest attempt at deriving a direct procedure for minimizing the arithmetic in a frontal solution appears to be that of King,[1] who was essentially concerned with obtaining efficient orderings for water distribution networks. It is interesting to note, however, that although King's algorithm is particulary suited to generating element numberings which minimize the maximum frontwidth, no mention of this is made in the original paper. Indeed, as published in its original form, the algorithm appears to be tailored for a solution procedure in which the variables are assembled and eliminated on a node-by-node basis. This is because the scheme concentrates on renumbering the nodes only.

To utilize King's technique in a frontal solver which interleaves the assembly and elimination operations on an element-by-element basis, it is necessary to relabel the elements so that the new nodal elimination order is preserved as closely as possible. This may be achieved by resequencing the elements in ascending order of their lowest-numbered nodes. An interesting aspect of this algorithm is that it attempts to select an ordering which minimizes the number of operations required to decompose the sparse matrix equations. To distinguish between different nodal numbering strategies, King[1] employs an ordering efficiency parameter, σ, which is defined by

$$\sigma = \sum_{i=1}^{N} M_i^2 \tag{12}$$

where M_i is the number of 'off-diagonal' nodes (i.e. current number of active nodes minus one) during the elimination of node i, and N is the total number of nodes.

The approximate theoretical basis for this criterion may be seen from equation (10). If the average value of w_i is relatively large, then we may write

$$T \simeq \tfrac{1}{2} \sum_{i=1}^{n} w_i^2 \tag{13}$$

Furthermore, if there is only 1 degree-of-freedom per node, then

$$T \simeq \tfrac{1}{2}\sigma = \tfrac{1}{2} \sum_{i=1}^{N} M_i^2 \tag{14}$$

The fundamental steps in King's algorithm may be summarized as follows:

1. Generate an adjacency list indicating the connectivity of all nodes in the grid. A node is said to be adjacent to another node if they share an element.
2. Select a starting node and relabel it as node one.
3. Generate a list of nodes which are currently active. Mark all references to these nodes in the adjacency list by negative values. Accumulate the ordering efficiency parameter, σ.
4. Examine each active node in the list and compute the increment in the number of active nodes if each of these nodes were to be eliminated. Note that a positive increment may occur only if the node is connected to positive entries in the adjacency list, and that the increment must be either a positive integer, zero or minus one.
5. Select and relabel the node which has the lowest increment in active nodes. In the case of a tie, choose the node which has been active the longest.
6. Delete the selected node from the list of active nodes, and add nodes which are adjacent to the selected node onto the list of active nodes (if they are not already active). In the adjacency list, mark all references to the newly added nodes by negative values.

7. Accumulate the ordering efficiency parameter, σ, by adding the square of the number of active nodes in the list.
8. Repeat steps 4–7 until all nodes have been relabelled.
9. Repeat steps 2–8 until all starting nodes have been processed.
10. Accept the numbering strategy which yields the smallest ordering efficiency parameter.

Steps 4 and 5 may be viewed as a criterion for ensuring that the frontwidth grows by a minimum amount, and are the essence of the King procedure. Note, however, that the algorithm only reorders the nodes and not the elements. As noted by Cuthill,[4] the King algorithm is quite efficient and, for finite element meshes with few nodes per element, will often furnish optimal or near-optimal frontwidths. A major disadvantage of the method, however, is that it is highly sensitive to the location of the starting node. For problems with a large number of nodes this is a severe drawback, since it is uneconomic to consider each node as a starting point.

Another heuristic method for resequencing finite element grids to reduce the maximum frontwidth has been given by Levy.[5] This method is very similar to that of King,[1] but is based on an expanded minimum front-growth principle. When searching for the next node to be relabelled at each stage, all nodes which have yet to be relabelled are considered. (Recall that in stage 4 of the King[1] algorithm, nodes are considered for possible relabelling only if they are currently in the front.) Furthermore, the criterion for assessing the merit of each numbering scheme is based simply on the smallest maximum nodal frontwidth.

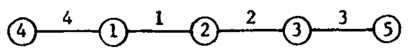
Because of the extended search which is conducted before selecting the next node to be renumbered the Levy algorithm is invariably slower than the King algorithm. It may, however, furnish maximum frontwidths which are closer to the true minimum.[4] As with King's method the starting nodes for the Levy scheme must be specified *a priori*. Due to the extra computation associated with the generation of each numbering sequence, this is again a serious limitation.

More recently Pina[6] has described another method for optimizing finite element numberings. As with the Levy and King algorithms, Pina's strategy is based on a minimum front-growth principle, but includes an additional search procedure which attempts to look ahead. The search refinement incorporated by Pina[6] is apparently useful in meshes comprised of elements with midside nodes. Later in this paper it will be shown that, for meshes which are comprised of a single type of high-order element, it is unnecessary to consider non-vertex nodes in the renumbering process. This is due to the fact that mesh numberings which are optimal for low-order elements are also optimal for high-order elements. As with the Levy and King techniques, Pina's method suffers from the disadvantage that the starting nodes must be specified by the user.

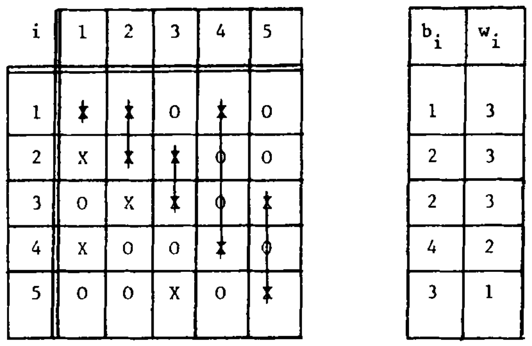
The indirect methods

When attempting to develop heuristic schemes for minimizing the frontwidth of a sparse set of matrix equations, it is fruitful to consider schemes which are aimed at minimizing the bandwidth. This is because the maximum frontwidth must always be less than, or equal to, the corresponding bandwidth (if the variables are eliminated in the same order). An illustration of this property is given in Figure 2 (taken from Cuthill[4]) for a simple grid of one-dimensional bar elements, each with 1 degree-of-freedom per node. The maximum bandwidth of a symmetric $n \times n$ matrix $[K]$ may be defined as

$$B = \max \{b_i : 1 \leq i \leq n\} \tag{15}$$



mesh of one-dimensional bar elements
(one degree of freedom per node)



i	1	2	3	4	5
1	X	X	0	X	0
2	X	X	X	0	0
3	0	X	X	0	X
4	X	0	0	X	0
5	0	0	X	0	X

b_i	w_i
1	3
2	3
2	3
4	2
3	1

Heavy lines denote active columns

$B = \max\{b_i : 1 \leqslant i \leqslant 5\} = 4$

$W = \max\{w_i : 1 \leqslant i \leqslant 5\} = 3$

Corresponding pattern of non-zero entries in global stiffness matrix [K]

Figure 2. Relationship between bandwidth and maximum frontwidth

where b_i is defined as the difference between $i+1$ and the column index of the first non-zero entry in row i of $[K]$. This definition of bandwidth differs from that given widely in the literature, since it includes the diagonal term.

The wavefront (or number of active equations) for row i, w_i, may be defined as the number of active columns in row i. A column j is said to be active if (i) $j \geqslant i$ and (ii) there is a non-zero element in that column with a row index k, such that $k \leqslant i$. The maximum frontwidth, or wavefront, is then defined by

$$W = \max\{w_i : 1 \leqslant i \leqslant n\} \tag{16}$$

From an inspection of Figure 2, it is clear that the maximum frontwidth, W, can never exceed the bandwidth B. Thus, one method of reducing the maximum frontwidth is first to resequence the nodes to minimize the bandwidth, and then to relabel the elements so that the new order of elimination is preserved as closely as possible. This approach has been used by Akin and Pardue[7] and, more recently, by Razzaque.[8] The effectiveness of this strategy is obviously dependent on the performance of the bandwidth minimization procedure, and thus suffers from the disadvantage of being indirect.

A NEW ELEMENT RESEQUENCING ALGORITHM

In this section a new element resequencing algorithm, which is suitable for use with a frontal solution programme, is described. The procedure first renumbers the nodes in an effort to obtain an optimum elimination sequence, and then uses this result to reorder the elements. This intermediate step is necessary because of the nature of the frontal method, where variables are assembled on an element-by-element basis but eliminated node by node. To renumber the nodes, a modified form of the King[1] algorithm is used. The main disadvantage associated with this technique, namely the difficulty of knowing where to initiate the renumbering process, is overcome by selecting the starting nodes automatically. As a result, only a few node-numbering sequences need to be generated. After choosing the nodal numbering scheme which yields the lowest maximum frontwidth, the elements are reordered in an ascending sequence of their (new) lowest-numbered nodes. This preserves the optimum elimination order as closely as possible. After the elements have been renumbered, the new node numbering scheme may be discarded.

Notation and definitions

As noted by Cuthill and McKee,[9] a finite element mesh may be treated as an undirected graph, with permutation of the rows and columns of the sparse matrix $[K]$ corresponding to a relabelling of the vertices of the graph. Some elements of graph theory prove useful in the development of heuristic reordering algorithms, and thus it is appropriate to state some essential definitions.

For the present purposes, a graph G is defined to be a pair $(N(G), E(G))$, where $N(G)$ is a non-empty finite set composed of members called *nodes* (cf. nodes in finite element terminology), and $E(G)$ is a finite set of unordered pairs composed of distinct members of $N(G)$ called *edges* (for a mesh of one-dimensional bar elements, an edge is equivalent to an element).† With reference to Figure 2, $N(G)$ is the set $\{1, 2, 3, 4, 5\}$, and $E(G)$ is the set $\{1, 2\}$, $\{2, 3\}$, $\{3, 5\}$ and $\{1, 4\}$. Note that for a grid of one-dimensional bar elements, the edge set $E(G)$ simply corresponds to the set of nodal definition vectors for the elements. A graph is said to be *undirected* if $E(G)$ is comprised of unordered pairs. The above definition of a graph excludes the occurrence of loops (i.e. edges which join nodes to themselves) and multiple edges (i.e. pairs of nodes which are connected by more than one edge).

The *degree* of a node i in G is defined as the number of edges incident to i. Two nodes i and j of a graph are said to be *adjacent* if there is an edge joining them (i.e. if there is an edge defined by the unordered pair $\{i, j\}$).

A *path* is defined by a sequence of edges. Any two *nodes* i and j of G are said to be *connected* if there is a path joining them such that consecutive edges share a common node. A *graph G* is said to be *connected* if each pair of distinct nodes is connected. Most graphs which correspond to finite element meshes are connected.

The *distance*, $d(i, j)$, between any two nodes i and j in a graph G is defined as the number of edges on the shortest path connecting them. A *diameter*, $D(G)$, of G is defined as the maximum possible distance between any pair of nodes, i.e.

$$D(G) = \max\{d(i, j): i, j \in N(G)\} \tag{17}$$

† Note that these edges should not be confused with the 'edges' of each finite element. In some instances they are equivalent, but not always, e.g. a single 4-noded quadrilateral generates a graph with six edges, four along its sides and two along its diagonals.

With reference to Figure 2, the distance between nodes one and three, for example, is two. The diameter, $D(G)$, of this graph is four, and the *endpoints* of the diameter are nodes four and five. Following George and Lui[10] a *pseudo-diameter*, $\delta(G)$, is defined by endpoints i and j for which $d(i, j)$ is *close* to $D(G)$. Nodes which define a pseudo-diameter are known as *pseudo-peripheral nodes*.

In bandwidth minimization algorithms which are based on graph theory (e.g. Cuthill and McKee,[9] Gibbs, Poole and Stockmeyer[11]) an important concept is the rooted level structure. A rooted level structure of a graph G is defined as a partitioning of the set of nodes $N(G)$ into levels $l_1(r)$, $l_2(r)$, ... $l_h(r)$ such that:

1. $l_1(r) = \{r\}$, where r is the root (or starting) node of the level structure.
2. All nodes adjacent to nodes in level $l_i(r)$: $1 < i < h$ are in levels $l_{i-1}(r)$, $l_i(r)$ and $l_{i+1}(r)$.
3. All nodes adjacent to nodes in level $l_h(r)$ are in levels $l_{h-1}(r)$ and $l_h(r)$.

Following George and Liu[10] the overall level structure may be expressed as the set $L(r) = \{l_1(r), l_2(r), \ldots l_h(r)\}$, where h is the depth of the level structure rooted at node r, and is simply the total number of levels. The width of level is defined by $|l_i(r)|$ (i.e. the number of nodes on level i), and the width of the level structure is defined by

$$\omega(r) = \max\{|l_i(r)|: 1 \leq i \leq h\} \tag{18}$$

With reference to Figure 2, the level structure rooted at node one is given by

$$L(1) = \{1, 2, 4, 3, 5\}$$

where $l_1(1) = \{1\}$, $l_2(1) = \{2, 4\}$, $l_3(1) = \{3\}$ and $l_4(1) = \{5\}$. For this example h is equal to four, and $\omega(1) = 2$.

Selection of starting nodes

When using the Cuthill–McKee algorithm for minimizing bandwidth, Gibbs, Poole and Stockmeyer[11] have observed that the best numbering schemes are often generated by starting at nodes which are endpoints of a diameter (or pseudo-diameter). This is because the bandwidth is related to the width of the level structure in this scheme, and these nodes tend to yield rooted level structures which are long and narrow. In the Cuthill–McKee procedure, a rooted level structure is assembled and relabelled by assigning consecutive integers to each level, considering one level at a time only; the resulting bandwidth must lie within the limits

$$\omega(r) + 1 \leq B \leq 2\omega(r) \tag{19}$$

where $\omega(r)$ is the width of the level structure rooted at node r and is defined by (18).

It is intuitively apparent that pseudo-peripheral nodes should also make good starting nodes for a heuristic algorithm which attempts to minimize the maximum frontwidth. For instance, with the King[1] scheme the front (of active nodes) will tend to propagate down the level structure in a manner such that the maximum level width provides an approximate measure of the maximum frontwidth. In the applications section of this paper, it is demonstrated that pseudo-peripheral nodes make excellent starting nodes for the King[1] algorithm.

An efficient method for locating a set of pseudo-peripheral nodes for a graph, G, is as follows:

1. Pick an arbitrary node, r, and generate the corresponding level structure $L(r) = \{l_1(r), l_2(r), \ldots l_h(r)\}$. Store the depth, h, and width, $\omega(r)$, for this level structure.

2. Generate the level structures for each node in level h of $L(r)$ (i.e. for all nodes which are at maximum distance from r). Select the node, s, which has the greatest level structure depth, h_s, and narrowest width $\omega(s)$. If $h_s > h$, set $r = s$, $h = h_s$, $\omega(r) = \omega(s)$ and go to step 1. If $h_s = h$ and $\omega(s) < \omega(r)$, set $r = s$, $\omega(r) = \omega(s)$ and go to step 1.
3. Store all entries on level $l_h(r)$ and the root node, r, to furnish the required set of pseudo-peripheral nodes.

Steps 1–3 are a modified version of the algorithm given by Gibbs, Poole and Stockmeyer.[11] Another algorithm for finding the endpoints of a pseudo-diameter has been given by George and Liu.[10] Typically, the above procedure furnishes the required set of starting nodes after only two or three iterations. In many cases the pseudo-diameter calculated is actually a true diameter, but there is no guarantee of this.

Node renumbering algorithm

Although the frontal method assembles the equations on an element-by-element basis, the variables are eliminated node by node. Thus, as an intermediate step towards reordering the elements, it is first necessary to ascertain an efficient elimination order for the nodes.

As described previously, there are essentially two different types of algorithms for renumbering nodes to reduce the maximum frontwidth. In the first of these, the nodes are renumbered using a minimum front-growth criterion, and are thus termed direct methods. In the second, the maximum frontwidth is reduced indirectly by minimizing the bandwidth, and uses the result that the maximum frontwidth must always be less than, or equal to, this quantity (providing the nodes are eliminated in the same order). If the Cuthill–McKee algorithm is used in the latter type of approach, it is interesting to note than an uppper bound on the maximum frontwidth is given by

$$W \leqslant 2\omega(r) \tag{20}$$

where $\omega(r)$ is the width of the level structure rooted at node r. In this paper, a direct numbering scheme is used. The fundamental steps are as follows:

1. Generate an adjacency list for each node $i: i \in N(G)$, noting that a node is said to be adjacent to another node if they share a common element. Compute and store the degree of each node i. This completely specifies the graph G.
2. Using the algorithm described in the previous subsection, compute a pseudo-diameter of the graph G, and assemble the associated set of pseudo-peripheral nodes. The latter constitute the set of possible starting nodes for steps 3–9.
3. Select a node from the list of pseudo-peripheral nodes and relabel it as node one. Assign node one an eliminated status. Using the adjacency list generated in step 1, mark all nodes which are adjacent to node one as currently active (i.e. in the current front). Store the number of active nodes.
4. Examine the nodes which are currently active, and calculate the increase/decrease in the number of active nodes if each of these nodes were to be eliminated. Note that the increment in the number of active nodes will be either a positive integer, zero, or minus one.
5. Select the node which, if eliminated, requires the smallest increase in the number of active nodes. In the case of a tie, select the node which has been active the longest. Relabel the chosen node, and assign it an eliminated status.
6. If the increment in active nodes $= -1$ go to step 7. Examine all nodes which are adjacent to the node just eliminated (relabelled) and mark them as being currently active. In this step, ignore nodes which already have an active or eliminated status.

7. Accumulate the number of active nodes. If the number of active nodes $\geq$ the maximum frontwidth from a previous scheme, abandon the current numbering strategy and go to step 3.
8. Repeat steps 4–7 until all nodes have been relabelled, and store the maximum frontwidth.
9. Repeat steps 3–8 until all entries in the pseudo-peripheral node list have been processed.

Incorporated in this algorithm is the minimum front-growth criterion due to King.[1] For some problems, the minimum front-growth criterion suggested by Levy[5] may furnish smaller fronts, but at the expense of additional computer time. To incorporate the Levy algorithm, step 4 needs to be modified as follows:

4. Examine all nodes which have yet to be relabelled, and calculate the increase/decrease in the number of active nodes if each of these nodes were be eliminated.

Both of these criteria may be included in one computer program. If the governing equations are to be assembled/eliminated many times, such as in nonlinear finite element applications, it may be worth while to spend the extra time and employ the Levy algorithm. On the other hand, if the equations are to be assembled/eliminated once only, it may be preferable to use the King scheme. For the latter case, renumbering is justified only if the saving in cost for one solution is greater than the cost associated with renumbering.

Element renumbering algorithm

In the most common form of frontal solution technique,[2,12] the assembly and elimination operations are interleaved on an element-by-element basis. Thus it is the order of the elements, not the nodes, which determines the efficiency of the solution procedure. In the algorithm described in the previous section, the nodes are renumbered in an attempt to minimize the number of active nodes during each stage of a node-by-node elimination procedure. This nodal reordering may be used to obtain an efficient front solution by relabelling the elements so that the node-by-node elimination order is (approximately) preserved. As noted by Akin and Pardue[7] and Razzaque,[8] this may be achieved by ordering the elements in ascending sequence of their lowest-numbered nodes. To illustrate the overall algorithm, a hand-worked example is given in Appendix I. A computer implementation, written in FORTRAN IV, is given in Appendix II.

Applications

In this section, the performance of the algorithm is assessed by applying it to a series of test problems. The range of finite element grids considered is illustrated in Figures 3–16.

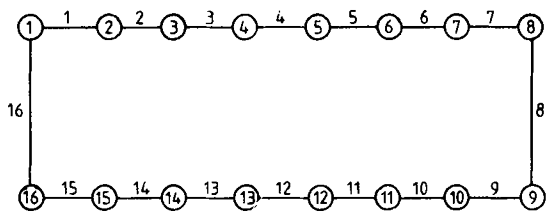
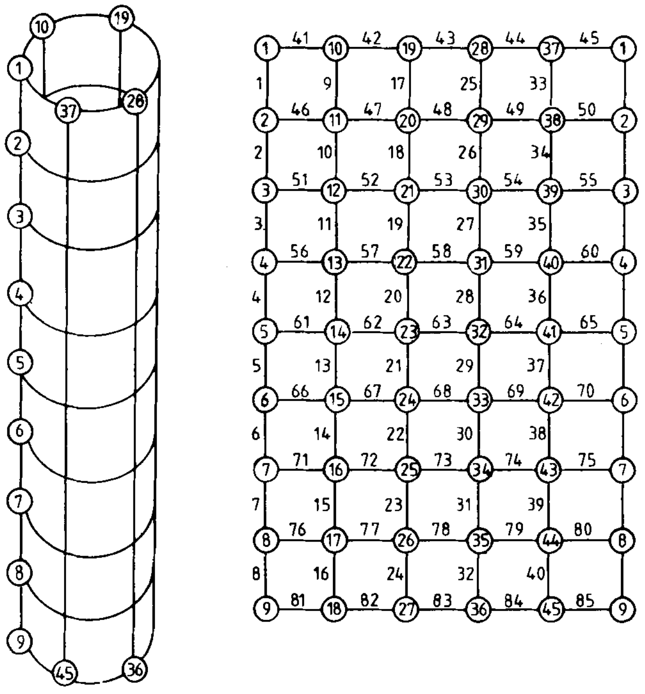


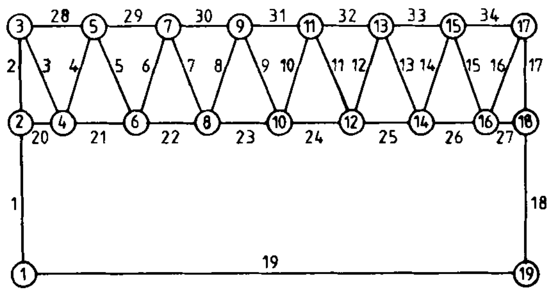
Figure 3. Example problem 1



85 bar elements
45 nodes

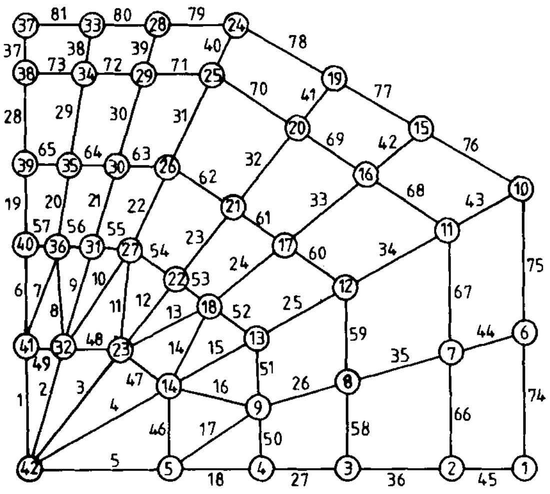
Figure 4. Example problem 2

Examples 1–4 have been taken from Cuthill and McKee,[9] and involve one-dimensional bar elements only. Examples 5 and 6, which involve meshes composed of 4-noded quadrilaterals and bars, are due to Grooms.[13] Example 7 depicts a water distribution network as described by King,[1] and examples 8 and 9 are continuum meshes taken from Akhras and Dhatt.[14] The remaining five examples are due to the authors, and are typical of grids which may arise when applying the finite element method to continuum problems.



34 bar elements
19 nodes

Figure 5. Example problem 3



81 bar elements
42 nodes

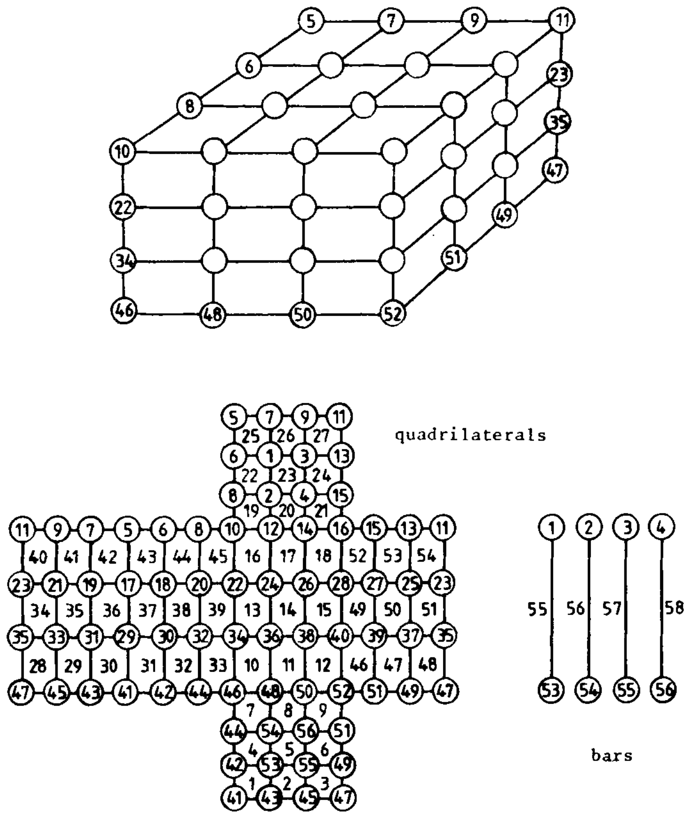
Figure 6. Example problem 4

Table I illustrates the maximum frontwidths for these problems, before and after renumbering, together with a comparison of the results presented by Razzaque.[8] In all cases where comparisons are available, the new algorithm yields the lowest, or equal lowest, maximum frontwidths. In examples 10 and 11, where 6-noded linear strain triangles are used to discretize

Table I. Results of frontwidth minimization algorithm for example problems

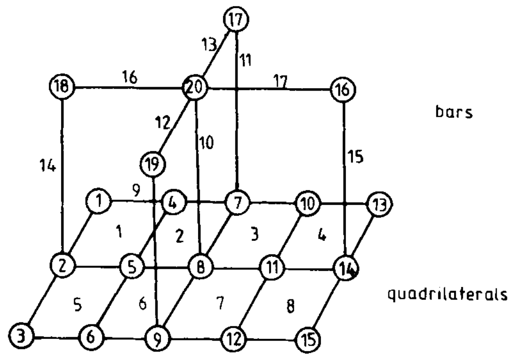
Example problem no.	Initial maximum frontwidth (nodes)	Final maximum frontwidth (nodes)	
		Present algorithm	Razzaque
1	3	3	3
2	45	6	7
3	19	4	4
4	41	7	8
5	35	15	21
6	8	7	7
7	11	6	—
8	23	18	23
9	30	14	14
10	28	12†	—
11	49	24†	—
12	43	31‡	—
13	23	11	—
14	26	19	—

† Meshes of 6-noded linear strain triangles—corner nodes only utilized in renumbering process, but frontwidth based on all nodes.
‡ Mesh of 15-noded cubic strain triangles—corner nodes only utilized in renumbering process, but frontwidth based on all nodes.



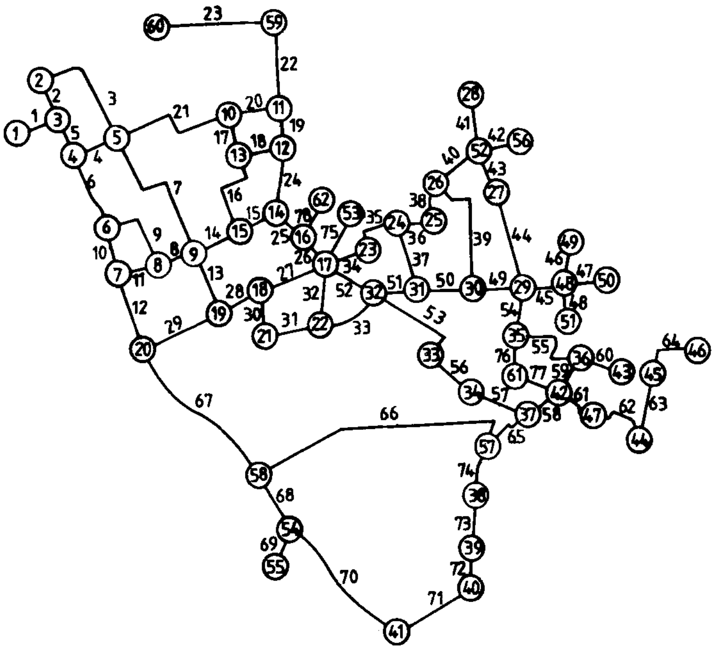
54 four-noded quadrilaterals
4 bar elements
56 nodes

Figure 7. Example problem 5



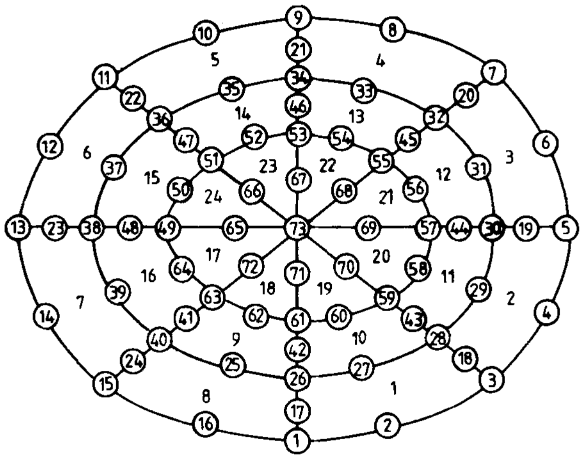
9 bar elements
8 four-noded quadrilaterals
20 nodes

Figure 8. Example problem 6



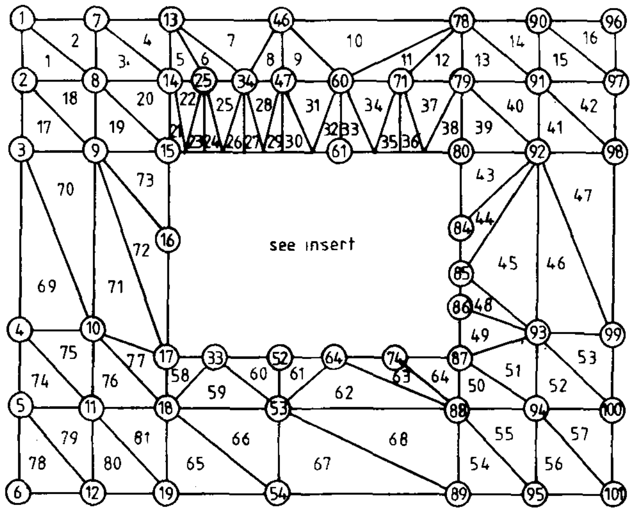
78 elements
62 nodes

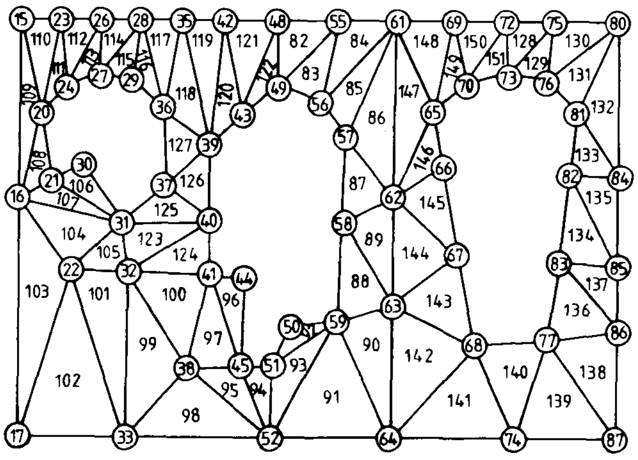
Figure 9. Example problem 7



16 eight-noded quadrilaterals
8 linear strain triangles
73 nodes

Figure 10. Example problem 8

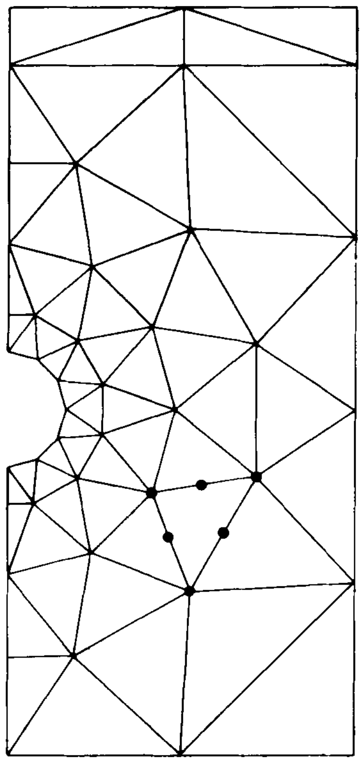




151 constant strain triangles
101 nodes

Figure 11. Example problem 9

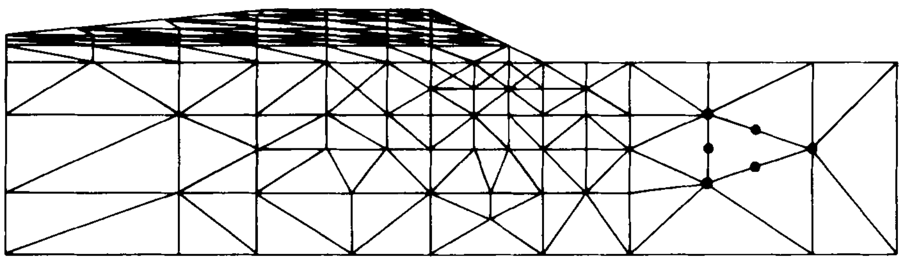
the domain, the elements have been renumbered by considering corner nodes only (i.e. by regarding each element as a 3-noded constant strain triangle). This procedure has also been used in example 12, for a grid of 15-noded cubic strain triangles. In general, for continuum meshes which are comprised of one type of high-order element, it is sufficient to consider the corner nodes only during the node and element relabelling process. This yields low maximum frontwidths and is inexpensive, since the number of corner nodes is often much less than the total number of nodes. For example, in the mesh of cubic strain trangles shown in Figure 14, only 32 nodes are involved in the node and element renumbering procedure, even though there are 413 nodes altogether.



58 linear strain triangles
141 nodes
42 corner nodes

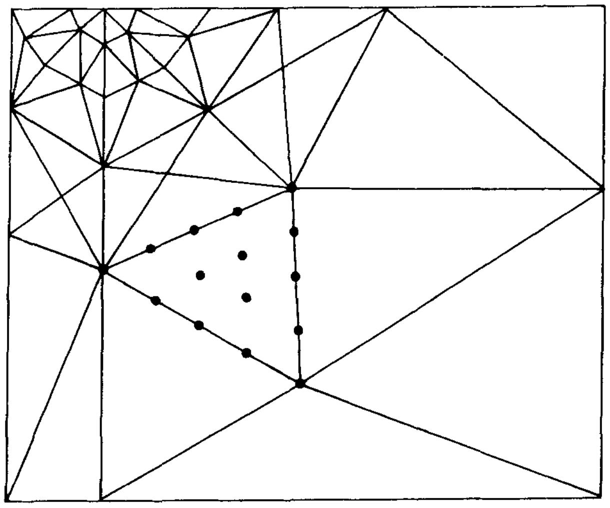
Figure 12. Example problem 10

Table II illustrates the computer times required to produce the element reorderings for each example. The statistics indicate the times required for assembling the nodal adjacency lists, locating the set of pseudo-peripheral starting nodes, and renumbering the nodes and elements. All of these results are for the IBM 370/165 installation at Cambridge, and were



190 linear strin triangles
415 nodes
113 corner nodes

Figure 13. Example problem 11



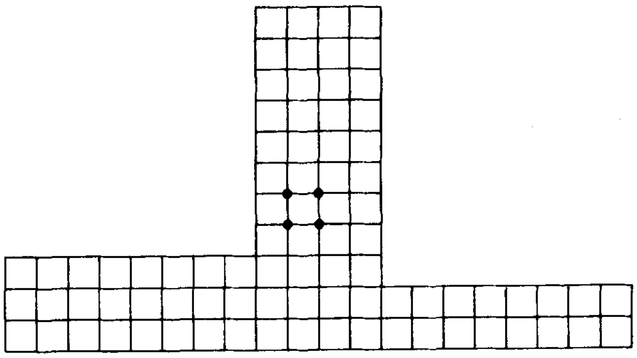
48 cubic strain triangles
413 nodes
32 corner nodes

Figure 14. Example problem 12

obtained using the internal clock of the machine for the optimising Q-compiler. The times quoted are accurate to the nearest one-hundredth of a second. In general, the selection of the starting nodes and the nodal renumbering procedure consume most of the time.

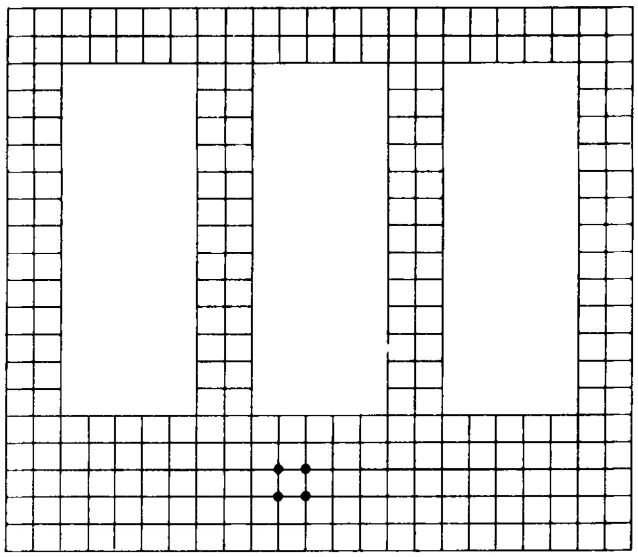
DISCUSSION

Generally speaking, the element reordering algorithm works best for meshes made up of triangular or one-dimensional elements. Indeed, for the cases presented with these types of



484 4-noded quadrilaterals
116 nodes

Figure 15. Example problem 13



265 4-noded quadrilaterals
360 nodes

Figure 16. Example problem 14

elements, it appears as though the automatic procedure yields numbering schemes which are quite close to the optimum. For example, Figure 12 illustrates a mesh which has been used by the soil mechanics group at Cambridge to study the behaviour of an unsupported tunnel. With this configuration of linear strain triangles, the best numbering scheme that could be produced by hand had a maximum frontwidth of 12 nodes.[15] This is identical to the maximum

Table II. Timing statistics for example problems (IBM 370/165, *Q*-compiler)

Example problem no.	Time (sec) Formation of adjacency list	Selection of starting nodes	Node renumbering	Element renumbering	Total
1	0·01	0·01	0·02	0·01	0·05
2	0·01	0·02	0·05	0·01	0·09
3	0·01	0·01	0·02	0·01	0·05
4	0·01	0·01	0·04	0·01	0·07
5	0·01	0·08	0·19	0·01	0·29
6	0·01	0·02	0·03	0·01	0·07
7	0·01	0·04	0·06	0·01	0·12
8	0·01	0·17	0·12	0·01	0·31
9	0·01	0·13	0·17	0·10	0·41
10	0·01	0·10	0·04	0·01	0·16
11	0·01	0·15	0·56	0·10	0·82
12	0·01	0·10	0·03	0·01	0·15
13	0·02	0·27	0·52	0·10	0·91
14	0·03	0·55	3·12	0·60	4·30

frontwidth achieved by the automatic algorithm. Similarly, Figure 13 shows an embankment mesh, also composed of linear strain triangles. For this problem, Britto[15] has derived a numbering scheme which has a maximum frontwidth of 20 nodes. This is slightly less than that obtained from the automatic scheme, which resulted in a numbering with a maximum frontwidth of 24 nodes.

When applied to grids with quadrilateral elements, the algorithm furnishes numbering schemes which are further from the optimum. This is because these meshes generate graphs in which many of the nodes have a high degree, and the effect of the tie-breaker becomes more pronounced (i.e. step 5 in the previous section). In addition, their pseudo-peripheral nodes often yield rooted level structures which are wide in comparison with their depth. An example of this is a square grid of 4-noded rectangles, with an equal number of elements along each side. The numbering scheme described in this paper is most effective when the pseudo-peripheral nodes give level structures which are long and narrow. Notwithstanding this shortcoming, the algorithm presented furnishes numberings which are efficient for a broad range of meshes comprised of quadrilateral elements.

CONCLUSIONS

An element reordering algorithm has been described which is suitable for finite element programmes using the frontal solution scheme. The algorithm, which first relables the nodes, and then the elements, has been shown to yield low maximum frontwidths for a wide variety of test meshes—including those which are composed of different types of elements. For grids of high-order elements of the same type, it has been demonstrated that efficient orderings may be obtained by considering the corner nodes only. This leads to significant savings in computational effort for these types of problems. Finally, for completeness, an illustrative FORTRAN IV implementation of the algorithm has been given.

ACKNOWLEDGEMENTS

The authors are indebted to several people for their assistance during the course of this work. The generosity of Dr. A. Britto in providing the data for some of the test examples, as well as continual discussion on the topic of renumbering finite element meshes, is gratefully acknowledged. Particular thanks are also due to Mr. M. J. Gunn and Mr. C. M. Szalwinski for their useful comments on the manuscript.

The first author was supported by an External Research Studentship from Trinity College while undertaking the reported research.

APPENDIX I: HAND-WORKED EXAMPLE

Consider the mesh of seven constant strain triangles shown in Figure 17. The graph, G, for this example is defined by the set of nodes $N(G)=\{1, 2, \ldots, 8\}$ and the set of edge pairs $E(G)=\{1, 2\}, \{1, 3\}, \{1, 4\}, \{1, 6\}, \{1, 7\}, \{2, 3\}, \{2, 6\}, \{3, 4\}, \{3, 5\}, \{4, 5\}, \{4, 7\}, \{4, 8\}, \{6, 7\}, \{7, 8\}$. In this case, the unordered pairs which define $E(G)$ correspond to the edges which define the elements. Figure 18 illustrates the algorithm for finding the starting nodes as described in the subsection entitled 'Selection of starting nodes'. For this example, the computed pseudo-diameter, $\delta(G)$, is identical to the real diameter, $D(G)$, and is equal to three. The resulting set of starting nodes is $\{5, 6\}$.

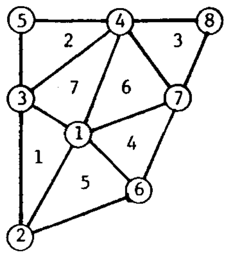


Figure 17. Finite element mesh of constant strain triangles

Before relabelling the elements, it is necessary to relabel the nodes. Consider the numbering scheme that results if node five is chosen as the starting node. Prior to the elimination of node five, the active nodes are {5, 3, 4}, and the number of active nodes is three. After elimination of node five, the active nodes are {3, 4}. If the King[1] algorithm is employed, then the next node to be eliminated must be either node three or node four. In order to eliminate node three, two new nodes (nodes one and two) need to be brought into the front. On the other

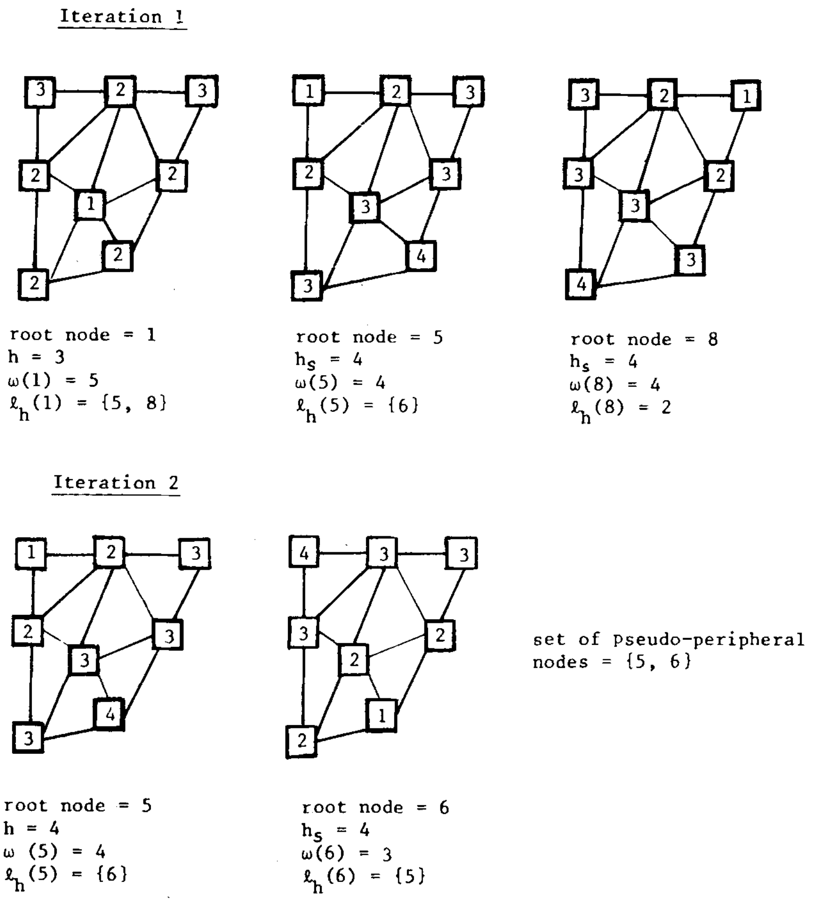


Figure 18. Algorithm for finding set of pseudo-peripheral nodes

Table III. Node renumbering algorithm

Eliminated nodes	Node to be eliminated	Nodes in front	Number of active nodes	New node no.
—	5	3, 4	3	1
5	3	4, 1, 2	4	2
5, 3	2	4, 1, 6	4	3
5, 3, 2	1	4, 6, 7	4	4
5, 3, 2, 1	6	4, 7	3	5
5, 3, 2, 1, 6	4	7, 8	3	6
5, 3, 2, 1, 6, 4	7	8	2	7
5, 3, 2, 1, 6, 4, 7	8	—	1	8

Table IV. Element renumbering algorithm

Element	Nodal definition vector	Lowest node no.	New element no.
1	2, 3, 4	2	2
2	1, 2, 6	1	1
3	6, 7, 8	6	7
4	4, 5, 7	4	5
5	3, 5, 4	3	4
6	4, 7, 6	4	6
7	2, 4, 6	2	3

hand, if node four is eliminated, three new nodes (nodes one, seven and eight) need to be activated. Therefore, the next node to be selected for elimination is node three, and the set of active nodes is {1, 2, 4}. This process may be repeated for the remaining nodes in the graph and is summarized in Table III. For this new nodal elimination order the maximum frontwidth is reduced from six to four. If node six is chosen to start the numbering scheme, the maximum frontwidth is again four.

After renumbering the nodes, the new nodal definition vectors for each element are as shown in Table IV. After reordering the elements in ascending sequence of their lowest-numbered nodes, the maximum frontwidth for element-by-element assembly is four, and the elimination order implied by the nodal renumbering is approximately preserved. The actual order of elimination, however, depends on where the variables are inserted into the front during the assembly phase.

APPENDIX II: DESCRIPTION OF FORTRAN IV IMPLEMENTATION

This appendix illustrates five subroutines, written in FORTRAN IV, which may be used to reorder the elements for a frontal solution package. The function of subroutine SETUP is to establish the adjacency list and degree of each node in the graph, G, of the finite element mesh. It is a modification of the code published by Collins,[16] and includes the facility for generating an adjacency list for the corner nodes only of a grid. The starting points for the node renumbering algorithm are determined using subroutines DIAM and LEVEL, which employ the algorithm described in the subsection entitled 'Selection of starting nodes' to compute a set of pseudo-peripheral nodes. Another algorithm for locating pseudo-peripheral nodes, together with its FORTRAN IV implementation, may be found in George and Liu.[10] In subroutine RESEQ1, the nodes are renumbered using the minimum front-growth principle described in the subsection entitled 'Node renumbering algorithm'. This new elimination order is then employed to relabel the elements, in ascending sequence of their lowest-numbered nodes, in subroutine RESEQ2.

A glossary of the essential variable names, in alphabetical order, is as follows:

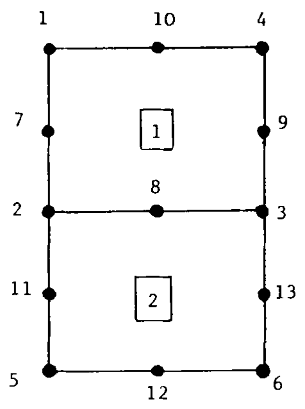
ALL	Logical variable which is used to ascertain whether all of the nodes in the finite element mesh are to be used in the reordering procedure. For meshes with one type of high-order element, it is necessary to consider the corner nodes only, and ALL is set to .FALSE. in the main driving routine. Otherwise, ALL is set to .TRUE.
IEN	Vector containing the initial element numbers. The address in this array indicates the new element numbers, e.g. IEN(6) = 1 means that the old number for new element six is one. Dimension equal to NET.
LEV	Vector containing level structure information. The level of node I is equal to LEV(I). Dimension equal to NODES.
MAXDEG	Control parameter indicating the maximum allowable degree of any node in the graph.
MAXNOD	Control parameter indicating the maximum allowable number of nodes for any element in the mesh.
MINMAX	New maximum frontwidth generated by the program. It should be set to a large value before entering subroutine RESEQ1. Note that for a mesh of high-order elements, where the corner nodes only are employed in the renumbering scheme, MINMAX does not represent the actual maximum frontwidth. Instead it represents the maximum frontwidth based on the corner nodes only. For grids of high-order elements in this case, the actual maximum frontwidth must be calculated in the normal manner, using the full list of nodes for each element and the new element numbering strategy (stored in array NEN).
NADJ	Vector containing the adjacency lists for all the nodes. Dimension equal to MAXDEG*NODES. The address list of nodes adjacent to node I is given by (I − 1)*MAXDEG + 1, (I − 1)*MAXDEG + 2, . . . , (I − 1)*MAXDEG + NDEG(I)
NDEG	Vector containing the degree of each node. Dimension equal to NODES. The degree of node I is equal to NDEG(I).
NEN	Vector containing the new element numbers. The address in this array indicates the old element number; e.g. NEN(1) = 6 means that the new number for old element one is six. Dimension equal to NET.
NET	Control parameter indicating the total number of elements in the mesh.
NEWNN	Vector containing the new node numbers generated for each starting node in subroutine RESEQ1. The address in this vector gives the old node number, e.g. NEWNN(1) = 6 means that the new number for old node one is six. Dimension equal to NODES.
NEWNUM	Vector containing the new node numbers which give the lowest maximum frontwidth in subroutine RESEQ1. The address in this vector gives the old node number, e.g. NEWNUM(1) = 6 means that the new number for old node

one is six. Dimension equal to NODES. After the new element numbers have been computed in subroutine RESEQ2, the contents of this array may be ignored and the original node numbers used.

NODES The number of nodes in the graph. In some cases this may differ from the total number of nodes in the finite element mesh; e.g. in a grid of one type of high-order element, NODES would be equal to the number of corner nodes. Note that if NODES is not equal to the total number of nodes, then the logical variable ALL must be set to .FALSE.

NPE Vector containing the number of nodes for each element. For element *I*, the number of nodes is equal to NPE(I). Dimension equal to NET.

NPN Vector containing the nodal definition vectors for all the elements. The addresses of nodes which define element *I* are given by (I − 1)*MAXNOD + 1, (I − 1)*MAXNOD + 2, ..., (I − 1)*MAXNOD + NPE(I). In this array, it is assumed that the corner nodes are listed first. Dimension equal to MAX-NOD*NET. For a mesh of high-order elements of a single type, where *N* corner nodes are used in the renumbering scheme, the corner nodes must be numbered from 1 to *N*. Figure 19 illustrates the form of the data required to assemble the NPN array.



NODES = number of corner nodes = 6

MAXNOD = maximum number of nodes per element = 8

NPN array for element 1

NPN(1),..... ,NPN(8)

$\underbrace{1 , 2 , 3 , 4}_{\text{corner nodes}} , \underbrace{7 , 8 , 9 , 10}_{\text{midside nodes}}$

NPN array for element 2

NPN(9),..... ,NPN(16)

$\underbrace{2 , 5 , 6 , 3}_{\text{corner nodes}} , \underbrace{11 , 12 , 13 , 8}_{\text{midside nodes}}$

Note that the 6 corner nodes must be numbered from 1 to 6 if the mesh is to be reordered using corner nodes only.

Figure 19. Data input for NPN array

NS The number of pseudo-peripheral nodes which are to be used as starting points for the node renumbering algorithm.

NSTART Vector of pseudo-peripheral nodes which are to be used as starting nodes for the node renumbering subroutine (RESEQ1). Maximum dimension of NODES.

NVN The number of corner nodes for each element for the case where the mesh is composed of elements of the same type. Used if the reordering process involves the corner nodes only.

```
      SUBROUTINE SETUP(NPN,NADJ,NDEG,NPE,NODES,NET,MAXNOD,MAXDEG,
     1NVN,ALL)
C*********************************************************************
C     SUBPROGRAM SETUP - COMPUTE ADJA CENCY LIST AND DEGREE FOR EACH
C                        NODE
C                      - MODIFIED VERSION OF COLLINS ROUTINE
C                      - USE ONLY CORNER NODES IF ALL=.FALSE.
C*********************************************************************
      LOGICAL ALL
      DIMENSION NPN(1),NADJ(1),NDEG(1),NPE(1)
C
      DO 10 J=1,NODES
   10 NDEG(J)=0
C
      DO 60 J=1,NET
      NN=NPE(J)
      IF(.NOT.ALL)NN=NVN
C
      DO 50 I=1,NN
      JNTI=NPN((J-1)*MAXNOD+I)
      JSUB=(JNTI-1)*MAXDEG
C
      DO 40 II=1,NN
      IF(II.EQ.I) GOTO 40
      JJT=NPN((J-1)*MAXNOD+II)
      MEM1=NDEG(JNTI)
      IF(MEM1.EQ.0) GOTO 30
C
      DO 20 III=1,MEM1
      IF(NADJ(JSUB+III).EQ.JJT) GOTO 40
   20 CONTINUE
C
   30 NDEG(JNTI)=NDEG(JNTI)+1
      NADJ(JSUB+NDEG(JNTI))=JJT
   40 CONTINUE
C
   50 CONTINUE
C
   60 CONTINUE
C
      RETURN
      END



      SUBROUTINE DIAM(NDEG,NSTART,LEV,NADJ,NODES,MAXDEG,NS)
C************************************************************
C     SUBPROGRAM DIAM - COMPUTE SET OF PSUEDO-PERIPHERAL NODES
C*****************************************************************
      DIMENSION NDEG(1),NADJ(1),LEV(1),NSTART(1)
      LOGICAL BETTER
C
C     BEGIN ITERATION
C     SELECT INITIAL ROOT NODE ARBITRARILY AND GENERATE ITS LEVEL
C     STRUCTURE
C
      IROOT=1
      ITER=0
   10 ITER=ITER+1
      CALL LEVEL(NDEG,LEV,IDEPTH,NADJ,IWIDTH,NODES,IROOT,MAXDEG)
C
C     CREATE LIST OF NODES WHICH ARE AT MAXIMUM DISTANCE FROM ROOT
C     NODE
C
      LHW=0
      DO 20 I=1,NODES
      IF(LEV(I).NE.IDEPTH)GOTO 20
      LHW=LHW+1
      NSTART(LHW)=I
   20 CONTINUE
C
C     STORE ROOT ON END OF LIST OF POSSIBLE STARTING NODES
C
      NS=LHW+1
      NSTART(NS)=IROOT
C
C     LOOP OVER NODES AT MAXIMUM DISTANCE FROM ROOT NODE
C     GENERATE LEVEL STRUCTURE FOR EACH NODE
C     SET SWITCH IF A LEVEL STRUCTURE OF GREATER DEPTH OCCURS
C
      BETTER=.FALSE.
      DO 30 I=1,LHW
      NEND=NSTART(I)
      CALL LEVEL(NDEG,LEV,NDEPTH,NADJ,NWIDTH,NODES,NEND,MAXDEG)
      IF(NDEPTH.LT.IDEPTH)GOTO 30
      IF((NDEPTH.EQ.IDEPTH).AND.(NWIDTH.GE.IWIDTH))GOTO 30
      IROOT=NEND
      IDEPTH=NDEPTH
      IWIDTH=NWIDTH
      BETTER=.TRUE.
   30 CONTINUE
      IF(BETTER)GOTO 10
C
      RETURN
      END


      SUBROUTINE LEVEL(NDEG,LEV,LSD,NADJ,MLW,NODES,NROOT,MAXDEG)
C*****************************************************************
C     SUBPROGRAM LEVEL - COMPUTE LEVEL STRUCTURE ROOTED AT NROOT
C*****************************************************************
      DIMENSION LEV(1),NDEG(1),NADJ(1)
C
C     INITIALISATION
C
      DO 10 I=1,NODES
   10 LEV(I)=0
      LEV(NROOT)=1
      KOUNT=1
      MLW=1
C
C     ASSIGN LEVELS TO VERTICES
C
      DO 40 L=2,NODES
      LW=0
C
      DO 30 I=1,NODES
      IF(LEV(I).GT.0)GOTO 30
      NCS=NDEG(I)
      JSUB=(I-1)*MAXDEG
C
      DO 20 JJ=1,NCS
      NODE=NADJ(JSUB+JJ)
      IF(LEV(NODE).NE.L-1)GOTO 20
      LSD=L
      LW=LW+1
      LEV(I)=L
      KOUNT=KOUNT+1
      IF(KOUNT.EQ.NODES)GOTO 50
      GOTO 30
   20 CONTINUE
C
   30 CONTINUE
      IF(LW.GT.MLW)MLW=LW
```

```
C
   40 CONTINUE
   50 IF(LW.GT.MLW)MLW=LW
C
      RETURN
C
      END



      SUBROUTINE RESEQ1(NADJ,NDEG,NEWNN,NEWNUM,NSTART,NODES,MAXDEG,
     1NS,MINMAX)
C*******************************************************************
C     SUBPROGRAM RESEQ1 - RESEQUENCE NODES FOR MINIMUM FRONTWIDTH
C*******************************************************************
      DIMENSION NADJ(1),NDEG(1),NEWNN(1),NEWNUM(1),NSTART(1)
C
C     LOOP OVER SET OF STARTING NODES
C
C
      DO 100 II=1,NS
      I=NSTART(II)
      DO 10 J=1,NODES
   10 NEWNN(J)=0
      NIF=NDEG(I)
      MAXFRT=NIF
      NEWNN(I)=1
C
C     NEGATE ALL NDEG ENTRIES FOR NODES WHICH ARE
C     ADJACENT TO STARTING NODE I
C
      NCN=NDEG(I)
      JSUB=(I-1)*MAXDEG
      DO 20 J=1,NCN
      N=NADJ(JSUB+J)
      NDEG(N)=-NDEG(N)
   20 CONTINUE
      NDEG(I)=-NDEG(I)
C
C     LOOP OVER NODES TO BE RENUMBERED
C
      DO 60 K=2,NODES
      MINNEW=10**10
      LMIN=10**10
C
C     LOOP OVER UNNUMBERED NODES
C     SKIP TO NEXT NODE IF OLD NODE IS ALREADY RENUMBERED
C     RESTRICT SEARCH TO ACTIVE NODES FOR KING SCHEME
C
      DO 40 J=1,NODES
      IF((NEWNN(J).GT.0).OR.(NDEG(J).GT.0))GOTO 40
      NEW=0
      MIN=10**10
      NCN=IABS(NDEG(J))
      LSUB=(J-1)*MAXDEG
C
C     COMPUTE THE INCREMENT IN ACTIVE NODES FOR EACH NODE J
C     COMPUTE WHEN NODE WAS FIRST ACTIVATED BY CHECKING FOR RENUMBERED
C     NEIGHBOURS WITH LOWEST NUMBERS
C
      DO 30 L=1,NCN
      N=NADJ(LSUB+L)
      IF(NDEG(N).GT.0)NEW=NEW+1
      IF(NEWNN(N).EQ.0)GOTO 30
      IF(NEWNN(N).LT.MIN)MIN=NEWNN(N)
   30 CONTINUE
C
C     SELECT NODE WITH SMALLEST INCREMENT IN ACTIVE NODES
```

```
C     IN THE CASE OF A TIE , SELECT NODE WHICH HAS BEEN ACTIVE THE
C     LONGEST
C
      IF(NDEG(J).LT.0)NEW=NEW-1
      IF(NEW.GT.MINNEW)GOTO 40
      IF((NEW.EQ.MINNEW).AND.(MIN.GE.LMIN))GOTO 40
      MINNEW=NEW
      LMIN=MIN
      NEXT=J
   40 CONTINUE
C
C     RENUMBER NODE AND COMPUTE NUMBER OF ACTIVE NODES
C     ABANDON SCHEME IF NUMBER OF ACTIVE NODES EXCEEDS PREVIOUS
C     LOWEST MAXIMUM FRONTWIDTH
C
      NEWNN(NEXT)=K
      NIF=NIF+MINNEW
      IF(NIF.GT.MAXFRT)MAXFRT=NIF
      IF(MAXFRT.GE.MINMAX)GOTO 80
C
C     NEGATE ALL NDEG ENTRIES FOR NODES WHICH ARE
C     ADJACENT TO NODE JUST RENUMBERED
C
      IF(MINNEW.EQ.-1)GOTO 60
      NCN=IABS(NDEG(NEXT))
      JSUB=(NEXT-1)*MAXDEG
      DO 50 J=1,NCN
      N=NADJ(JSUB+J)
      IF(NDEG(N).GT.0)NDEG(N)=-NDEG(N)
   50 CONTINUE
C
   60 CONTINUE
C
C     STORE NUMBERING SCHEME GENERATED
C     RESET NDEG TO POSITIVE VALUES
C
      DO 70 J=1,NODES
   70 NEWNUM(J)=NEWNN(J)
      MINMAX=MAXFRT
   80 DO 90 J=1,NODES
   90 NDEG(J)=IABS(NDEG(J))
  100 CONTINUE
      MINMAX=MINMAX+1
C
      RETURN
      END



      SUBROUTINE RESEQ2(NEWNUM,NPN,NEN,IEN,NPE,MAXNOD,NET,NODES,NVN,ALL)
C*********************************************************************
C     SUBPROGRAM RESEQ2 - RESEQUENCE ELEMENT NUMBERS TO MINIMISE
C                         THE FRONTWIDTH
C                       - REORDER THE ELEMENTS IN AN ASCENDING SEQUENCE
C                         OF THEIR LOWEST NUMBERED NODES
C*********************************************************************
      DIMENSION NEWNUM(1),NPN(1),NEN(1),NPE(1),IEN(1)
      LOGICAL ALL
C
      DO 10 I=1,NET
   10 NEN(I)=0
      KOUNT=0
C
C     LOOP OVER EACH NEW NODE NUMBER
C     LOOP ONLY OVER CORNER NODES IF ALL=.FALSE.
C
      DO 40 I=1,NODES
```

```
C
C     LOOP OVER EACH ELEMENT
C     SKIP TO NEXT ELEMENT IF ALREADY RENUMBERED
C
      DO 30 J=1,NET
      IF(NEN(J).GT.0)GOTO 30
      NN=NPE(J)
      IF(.NOT.ALL)NN=NVN
      I1=(J-1)*MAXNOD
C
C     LOOP OVER EACH NODE IN ELEMENT
C     USE ONLY CORNER NODES IF ALL=.FALSE.
C     ASSUMED THAT CORNER NODES ARE LISTED FIRST IN NODAL DEFINITION
C     VECTORS IF ALL=.FALSE.
C
      DO 20 K=1,NN
      N=NPN(I1+K)
      N=NEWNUM(N)
      IF(N.NE.I)GOTO 20
      KOUNT=KOUNT+1
      NEN(J)=KOUNT
      IEN(KOUNT)=J
      IF(KOUNT.EQ.NET)GOTO 50
      GOTO 30
   20 CONTINUE
C
   30 CONTINUE
C
   40 CONTINUE
C
   50 RETURN
      END
```

REFERENCES

1. I. P. King, 'An automatic reordering scheme for simultaneous equations derived from network systems', *Int. J. num. Meth. Engng*, **2**, 523–533 (1970).
2. B. M. Irons and S. Ahmad, *Techniques of Finite Elements*, Ellis Horwood, Chichester, U.K., 1980.
3. E. Hinton and D. R. J. Owen, *Finite Element Programming*, Series in Computational Mathematics and Applications, vol. 1, Academic Press, London, 1977.
4. E. Cuthill, 'Several strategies for reducing the bandwidth of matrices', in *Sparse Matrices and Their Applications* (Rose, D. J. and Willoughby, R. A., Eds.), Plenum Press, New York, 1972.
5. R. Levy, 'Resequencing of the structural stiffness matrix to improve computational efficiency', *Jet Propul. Lab. Q. Tech. Rev.*, **1**, 61–70 (1971).
6. H. L. G. Pina, 'An algorithm for frontwidth reduction', *Int. J. num. Meth. Engng*, **17**, 1539–1546 (1981).
7. J. E. Akin and R. M. Pardue, 'Element resequencing for frontal solutions', in *The Mathematics of Finite Elements and Applications* (*MAFELAP* 1975) (Whiteman, J. R., Ed.), Academic Press, London, 1975, pp. 535-541.
8. A. Razzaque, 'Automatic reduction of frontwidth for finite element analysis', *Int. J. num. Meth. Engng*, **15**, 1315–1324 (1980).
9. E. Cuthill and J. McKee, 'Reducing the bandwidth of sparse symmetric matrices', *Proc. A.C.M. Nat. Conf.*, Association for Computing Machinery, New York, 1969.
10. A. George and J. W. H. Liu, 'An implementation of a pseudo-peripheral node finder', *A.C.M. Trans. Math. Software*, **5**, 284–295 (1979).
11. 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.* **2**, 236–250 (1976).
12. B. M. Irons, 'A frontal solution program for finite element analysis', *Int. J. num. Meth. Engng*, **2**, 5–32 (1970).
13. H. R. Grooms, 'Algorithm for matrix bandwidth reduction', *J. Struct. Div., A.S.C.E.* **98**(ST1), 203–214 (1972).
14. G. Akhras and G. Dhatt, 'An automatic node relabelling scheme for minimising a matrix or network bandwidth', *Int. J. Num. Meth. Engng*, **10**, 787–797 (1976).
15. A. M. Britto, private communication.
16. R. J. Collins, 'Bandwidth reduction by automatic renumbering', *Int. J. num. Meth. Engng*, **6**, 345–356 (1973).