MEMORY-ECONOMIC

FINITE ELEMENT AND NODE RENUMBERING

by

© Hesham A. Auda; B.Sc. Cairo University

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Engineering

Department of Electrical Engineering
McGill University, Montreal, Canada
August 1981.

MEMORY-ECONOMIC
FINITE ELEMENT AND NODE RENUMBERING

by

Hesham A. Auda
Department of Electrical Engineering

McGill University, Montreal, Canada
August 1981

To my parents;

and to my brother Gehad.

Abstract

In this thesis, a memory-economic finite element renumbering strategy for frontal solutions is introduced. The new strategy is based on constructing an element level structure of maximal depth within frontal storage limitations. The memory-resident elements at any stage of the renumbering procedure are exactly the same as the elements of the active set in the frontal procedure. $O((m*N)^2)$ comparisons are required to construct such a level structure, where m is the maximum number of nodes per element and N is the number of elements in the mesh.

Extensive experimental testing of the new algorithm is carried out on a single user minicomputer. The test problems employed range in size from the relatively small to those containing approximately 1000 elements, with different topological structures. The experimental results obtained indicate that the new algorithm has a reliable performance and great memory saving ability.

Résumé

Cette thèse présente une nouvelle stratégie de réénumération des éléments, très économique en mémoire, qui prépare les systèmes d'équations rencontrés dans les problèmes d'éléments finis à être résolus par solution frontale. Cette nouvelle stratégie ordonne les éléments en une structure en niveaux, de profondeur maximale, tout en respectant les restriction en mémoire imposeés par la méthode de solution. Les éléments résidents en mémoire à n'importe quelle étape de la réénumération sont les mêmes que les éléments actifs lors de la résolution par méthode frontale. Le nombre de comparaisons requises, pour construire une telle structure est de l'ordre $O((m*N)^2)$, où m est le nombre maximum de noeuds par élément et N le nombre d'éléments dans le problème.

Le nouvel algorithme est éprouvé avec un ordinateur à usage personnel. Plusieurs problèmes, qui vont de petits à d'autres comprenant jusqu'à 1000 éléments, ont étés résolus pour différentes topologies. Les expériences démontrent de grandes économies en mémoire et la haute fiabilité de ce nouvel algorithme.

Acknowledgements

I would like first to thank my research supervisors Drs P.P. Silvester and Z.J. Csendes -- Dr. Silvester for introducing me to the subject and giving guidance at the early stages of the work and Dr. Csendes for his continual encouragement and numerous constructive suggestions that helped to keep this thesis in shape.

My thanks also go to Dr. D.A. Lowther for his helpful comments on the thesis; to M. J.P. Vandelac for his fine translation of the abstract and the Bullet model he provided me for use in my experiments; and to Mlle J. Lelièvre for her perfect typing of the tables included in this thesis.

Table of Contents

Chapter 1

Introduction

Solution of continuum problems by the finite element method involves replacing the differential or integral equations that describe a physical field, by a set of simultaneous algebraic equations [4]. This system of equations is usually very sparse, and many ingenious techniques have been developed to take advantage of the sparsity so as to allow solution of large systems of equations within reasonable limitations of memory and computer time.

Two essentially distinct, though related, ways exist for solving finite element equations while making good use of sparsity. The first, and well established, viewpoint proceeds by noting that in the solution of equations by Gaussian elimination, fill-in cannot occur to the left of the leftmost nonzero matrix element in any row. This observation leads directly to the popular band-matrix and profile-storage algorithms [15]. However, the success of these algorithms depends critically on the order of node numbering in the finite element mesh. For this reason, a considerable amount of effort has been expended on developing methods for bandwidth or profile minimization by renumbering the nodes in the mesh.

A widely used method is that first developed by Cuthill and McKee [7] and subsequently applied to finite element problems by George [10] in a mildly modified form. The methods of Collins [5] and King [19] are closely related to the Cuthill-McKee technique, and share its weaknesses: none of these methods is capable of generating its own starting node for renumbering. This problem was largely overcome by Gibbs, Poole and Stookmeyer [16], who showed how to determine a pair of nodes which are at maximum, or near maximum, distance apart in the graph corresponding to the coefficient matrix, and thereby to identify two good possible starting nodes.

A second viewpoint is that first propounded by Irons [18]: sparsity of the coefficient matrix results from the fact that all variables that occur in a given finite element are strongly connected (many nonzero matrix entries), while variables occuring in different elements are weakly connected (few nonzero matrix entries) if at all. It therefore appears useful to concentrate attention on the finite element structure rather than on the node incidence graph. This view leads directly to the now quite widespread frontal solution technique. This approach has been generalized and much more fully developed by George [11], as the so-called nested dissection method.

The essential principle in the frontal procedure is to produce the triangular factors of the coefficient matrix

directly, without explicitly writing out the matrix. To do this, matrix assembly and factorization of the system of equations are done in tandem so that only a small portion of the matrix needs to be memory-resident at any stage of the process. This makes it possible to solve even large sets of equations on small computers. However, success in this approach depends solely on arranging the elements in such a sequence that memory storage is minimized. For this reason, a similar effort, though less, has been expended on developing methods to find good element sequences.

The introduction of virtual-memory operating systems with very large addressing spaces has led to great ease in manipulating large matrices, but has not in any way removed the need for good frontal algorithms. Virtual memory management makes it possible to define and access very large matrices in a random fashion, but exacts its price in the form of paging between random-access (real) memory and a secondary storage system (disc). Thus even in a virtual memory system, algorithms for ordering elements remain essential if extreme numbers of paging operations are to be avoided.

Unfortunately, existing element sequencing algorithms [2,20] have been greatly influenced by the methods used for node renumbering; some of them [2,25] even use node renumbering algorithms as a necessary primary step. As a result, they possess an important disadvantage: their memory

demands correspond to the full set of elements and/or nodes in the finite element mesh. For large problems, this is prohibitively large. The ironic situation then arises where methods requiring large computing environment are used to find element sequences for a subsequent small-core frontal solution.

In this thesis, we develop a method for finding element sequences for frontal solutions, which requires storage amounting to the maximum size of the active element set at any stage of the frontal procedure. These sets which may be initially quite large, get reduced to a maximum size of O(sqrt(N)), where N is the number of elements in the mesh. The algorithm presented arises naturally by considering finite element renumbering methods from a frontal perspective. Surprisingly, the problem of finding element sequences for frontal solutions from a minimum (real) memory point of view has not been addressed before, despite its practical importance.

The outline of the thesis is as follows: in chapter 2, we briefly review the frontal solution procedure for finite element systems of equations. In chapter 3, we consider the renumbering problem for such solutions; and in chapter 4, we give a fairly detailed description of the new strategy. As is the case with renumbering algorithms, we evaluate the new algorithm in chapter 5 through extensive experimental tests. Chapter 6 includes the conclusion of the work in the thesis.

The basic matrix notations and definitions used throughout the thesis are given in appendix A.

Chapter 2

The Frontal Solution

Finite element equations are usually assembled on an element-by-element, rather than node-by-node, basis [6,26]. Element-oriented frontal solutions [18] are therefore natural to finite element problems. Only a brief account of the frontal procedure is given below, a more complete description may be found elsewhere [1].

The essential principle in the frontal procedure is to produce the triangular factors of the coefficient matrix directly, without explicitly writing out the matrix. At any given moment during the frontal procedure, the finite elements of a problem may be classified into three sets:

F : Elements associated with matrix rows and columns that have been already processed, i.e., decomposed and written to a file;

A : Elements associated with at least some matrix rows and columns that have been partly assembled, but have not yet been fully processed; and

E : Elements associated with only matrix rows and columns whose numerical processing has not yet begun.

The set of variables connecting these sets is called the front; the maximum size it attains is the frontwidth. The application of the frontal procedure on the finite element mesh in figure 2.1 is demonstrated below.

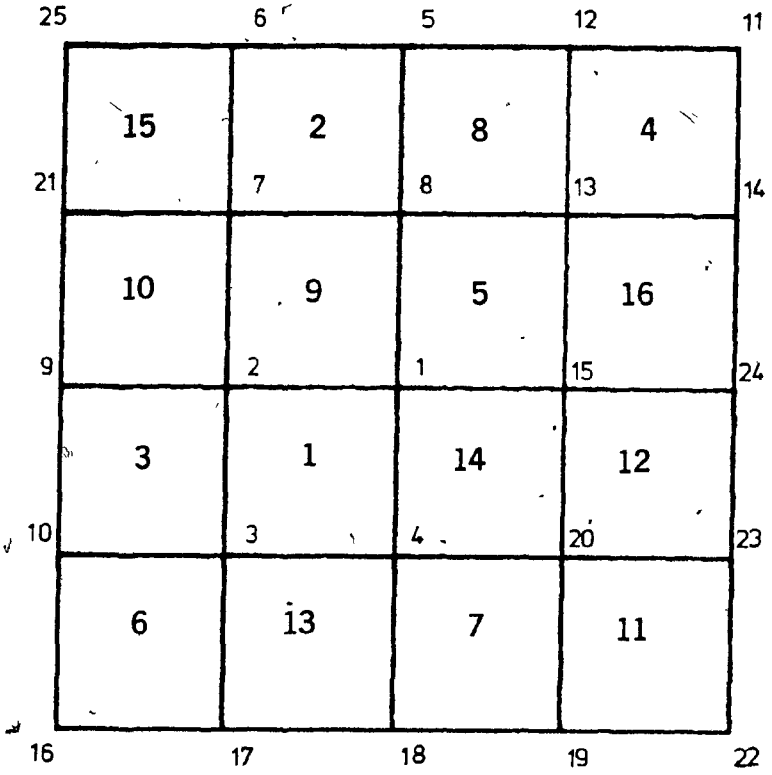


Fig 2.1 16 first order quadrilateral finite elements in a square.

The Application of the Frontal Procedure on the Finite Element Mesh in Figure 2.1

	F	A	|A|	E	Front	|Fr|
1		1,3,5,6,7, 9,10,13,14	9	2,4,8, 11, 12,15,16	1,2,3,4	4
2		1,3,5,6,7, 9,10,13,14 ,2,8,15	12	4,11,12,16	1,2,3,4,5, 6,7,8	8
3		1,3,5,6,7, 9,10,13,14 ,2,8,15	12	4,11,12,16	1,2,3,4,5, 6,7,8,9,10	10
4	4	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16	14	11,12	1,2,3,4,5, 6,7,8,9,10 ,12,13,14	13
5	4	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16,12	15	11	1,2,3,4,5, 6,7,8,9,10 ,12,13,14, 15	14
6	4,6,3	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16,12	15	11	1,2,3,4,5, 6,7,8,9,12 ,13,14,15, 17	14
7	4,6,3	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16,12,11	16		1,2,3,4,5, 6,7,8,9,12 ,13,14,15, 17,18, 19, 20	17
8	4,6,3,8,2	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16,12,11	16		1,2,3,4,6, 7,8,9, 13, 14,15, 17, 18,19,20	15

	F	A	|A|	E	Front	|Fr|
9	4,6,3,8,2, 9,5	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16,12,11	16		1,2,3,4,6, 7,9,13,14, 15,17, 18, 19,20	14
10	4,6,3,8,2, 9,5,10,1	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16,12,11	16		1,3,4,6,7, 13,14, 15, 17,18, 19, 20,21	13
11	4,6,3,8,2, 9,5,10, 1, 11,7	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16,12,11	16		1,3,4,6,7, 13,14, 15, 17,18, 20, 21,23	13
12	4,6,3,8,2, 9,5,10, 1, 11,7,12	1,3,5,6,7, 9,10,13,14 ,2,8,15,4, 16,12,11	16		1,3,4,6,7, 13,14, 15, 17,18, 20, 21,24	13
13	4,6,3,8,2, 9,5,10, 1, 11,7,12,13	1,5,7,9,10 ,13,14,2,8 ,15,4, 16, 12,11	14		1,4,6,7,13 ,14,15,20, 21,24	10
14	4,6,3,8,2, 9,5,10, 1, 11,7,12,13 ,14	5,9,10,14, 2,8,15, 4, 16,12	10		6,7,13,14, 15,21,24	7
15	4,6,3,8,2, 9,5,10, 1, 11,7,12,13 14,15	5,14,8, 4, 16,12	6		13,14, 15, 24	4
16	4,6,3,8,2, 9,5,10, 1, 11,7,12,13 ,14,15,16					

Evidently, only that part of the matrix corresponding to the front variables needs to be memory-resident. It is therefore necessary to know when all of the work with some variables is terminated to allow for front contraction. Although destination allotment is completely arbitrary, it is usual to assign to each variable its longevity, i.e, the span of elements for which it remains active (in the front). Thus, in the above demonstration, the variable number 1, for example, will have element number 14 as its destination. Destinations may alternatively be assigned to the elements rather than to the nodes. In this case, element number 14 will be the destination for elements 1, 7, 11 and 13; and the variables 1, 4, 18, 19, 20 and 21 may then be eliminated from the front altogether. Indeed, the whole task may be regarded as starting with all the elements belonging to the initial set E, and continuing until all have been relegated to the final set F, keeping at all times only the active portion of the matrix in fast core.

Frontal solutions of finite element problems are quite diverse. They include, among others, direct solution techniques for systems of linear equations [18] and the subspace iteration method for the generalized eignvalue problem [3]. Application of the frontal concept to the practical Manteuffel algorithm for stablizing incomplete Choleski decompositions [27] to allow solution of finite element field problems by the preconditioned conjugate gradients method is now an area of active research [23].

Chapter 3

Renumbering for Frontal Solutions

In this chapter, we consider the renumbering problem for frontal solution procedures of finite element systems of equations. The success of frontal solutions is shown to depend solely on the order in which the elements, and not the nodes, are eliminated. A criterion for finding element sequences for these solutions is given. The convenience of the element level structure as a vehicle in renumbering for frontal solutions is also established.

3.1 Numbering Requirements of Frontal Solutions

The success of frontal solutions clearly hinges on reducing the frontwidth so that fast core storage is minimized. For a given mesh topology, the front associated with a particular element depends only on the order in which the elements are eliminated. Frontwidth reduction can be accomplished, as the simple demonstration in chapter 2 readily suggests, by arranging the elements in such a sequence that only relatively few elements belong to the active set A at any stage of the frontal procedure.

Since adjacent elements are interconnected, minimizing the size of set A is equivalent to minimizing the difference in sequential numbers of adjacent elements. In other words,

exactly the same topological criteria apply to the frontal solution as apply to the standard Gaussian elimination, except for the fundamental difference that it is the element numbering, not the node numbering, which is of prime importance. Even a truly optimal node numbering does not guarantee the success of the frontal solution. This is immediately seen from the demonstration in the previous chapter: no matter what the node numbers are, the front sizes remain unchanged.

Nodes may only be renumbered after the elements are renumbered if at all. In fact, node renumbering is considered irrelevant to the frontal procedure, and the node numbers are sometimes termed "nicknames" to emphasize this fact [18]. However, a good node numbering is still required for more efficient computations, but as will be seen in section 3.3, a reasonably good node numbering is automatically induced by a good element numbering.

3.2 Element Renumbering for Frontal Solutions

Element renumbering strategies for frontal solutions may be classified as direct or indirect. Direct strategies attempt to find an element sequence without renumbering the nodes first. On the other hand, indirect strategies proceed by first renumbering the nodes so that minimum bandwidth is obtained, and then renumber the elements according to the new node pattern. Two direct and two indirect element

renumbering strategies are described below.

The first direct strategy is due to Akin and Pardue [2]. In their procedure, an element of minimum degree (the number of adjacent elements) is identified and renumbered first. The elements adjacent to the first element are then renumbered in increasing order of their current degrees (the number of adjacent elements yet to be renumbered). The un-renumbered elements adjacent to each new element in sequence (said to form an element level) are next renumbered in a similar fashion. The process is continued, level by level, until all the elements have been renumbered. An element sequence for the model mesh in figure 2.1 could be {15, 2, 10, 9, 8, 5, 3, 1, 14, 4, 16, 12, 6, 13, 7, 11}.

The second direct strategy has been given by Liu [20], and is based on the observation that minimizing the number of side-connections between eliminated and remaining elements at any stage of the frontal procedure can lead to a frontwidth reduction. An implementation of Liu's strategy works on the "dual graph" of the finite element mesh, where each element is represented by a node, and any two of these nodes are connected (by an edge) if and only if their corresponding elements share a common side. Figure 3.1 shows the dual graph of the mesh in figure 2.1.

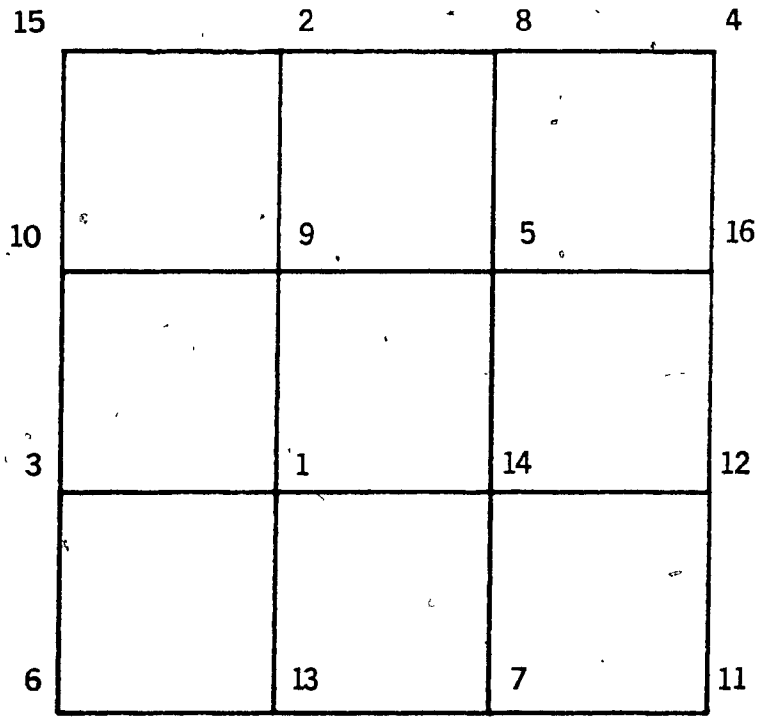


Fig 3.1 The dual graph of the finite element mesh in figure 2.1.

The set of edges connecting the renumbered and un-renumbered nodes of the dual graph is called the edge front; its size, the edge frontwidth, is now the quantity to be minimized. An element sequence is induced as follows: at each step, the nodes connected to the already renumberd ones are examined, and the node (element) increasing the edge frontwidth the least is renumbered next. An element sequence for the mesh in figure 2.1 could be {15, 2, 8, 4, 10, 9, 5, 16, 3, 1, 14, 12, 6, 13, 7, 11}.

The two indirect element renumbering strategies we now

describe are essentially similar. The first indirect strategy is also due to Akin and Pardue [2]. The Cuthill-McKee algorithm is initially used to renumber the nodes for a minimum bandwidth. The new node pattern is then utilized for renumbering the elements as follows: for each new node in sequence, the un-renumbered elements containing the new node are renumbered in increasing order of their current degrees. A possible element and node numbering obtained using this procedure for the mesh in figure 2.1 is shown in figure 3.2.

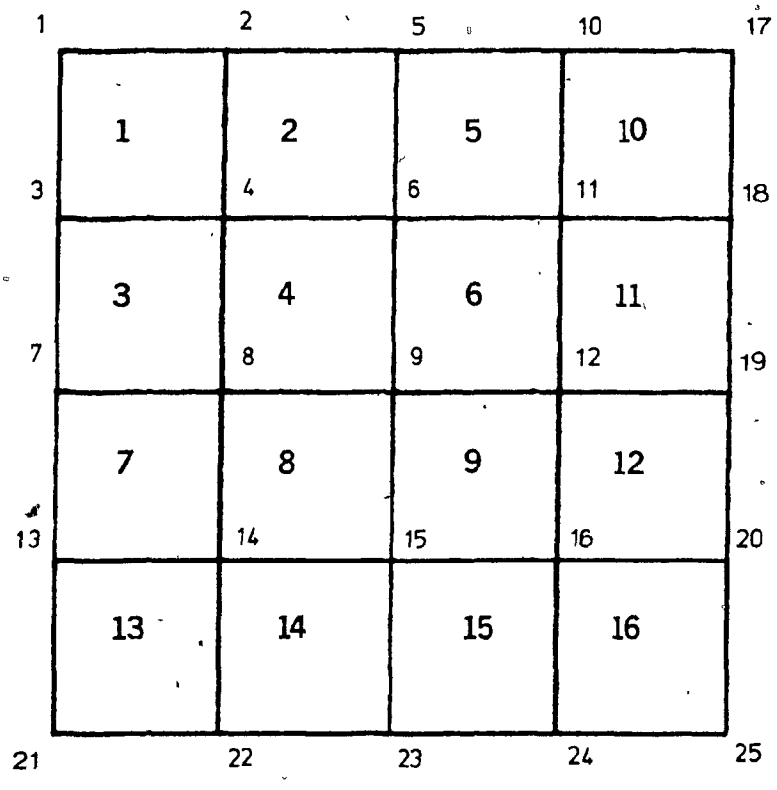


Fig 3.2 An element and node numbering for the mesh in figure 2.1 obtained by the Cuthill-McKee-Akin-Pardue indirect procedure.

The second indirect strategy has been recently published by Razzaque [25]. Again, the nodes are first renumbered for a minimum bandwidth, and the elements are then renumbered accordingly. However, element renumbering is now a little different: they are renumbered in increasing order of their least nodes. If the Cuthill-McKee algorithm is used for node renumbering, a possible element and node numbering for the mesh in figure 2.1 obtained by Razzaque's procedure would be similar to that given in figure 3.2 with the Akin-Pardue indirect procedure.

3.3 Node Renumbering for Frontal Solutions

Node renumbering for frontal solutions is not as crucial as element renumbering. In this section, we show that a good node numbering is automatically induced by a good element numbering. Two different techniques are described below.

Let the elements of a mesh be renumbered by the Akin-Pardue direct procedure. Let the nodes be renumbered after that as they occur, element by element in sequence (straight node renumbering). The fact that nodes in a single finite element are interconnected [10] ensures that nodes renumbered this way will have a Cuthill-McKee-like pattern. Figure 3.3 shows an element and node numbering obtained using this procedure for the mesh in figure 2.1.

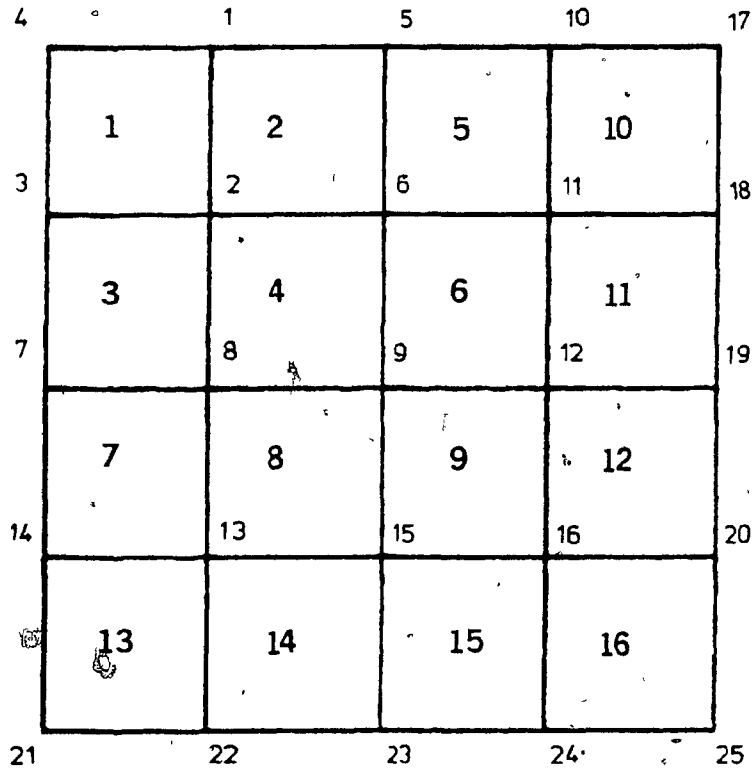


Fig 3.3 An element and node numbering for the mesh in figure 2.1 obtained by the Akin-Pardue procedure, and an element-by-element straight node renumbering.

Another node renumbering strategy has been suggested by Liu [20]: for a given element sequence, the nodes are renumbered as they are removed from the mesh by element elimination. Liu has developed much theory to prove that a minimum profile node numbering for finite element systems of equations could be obtained in this way. A combination of Liu's element and node numbering strategies gives the element and node numbering shown in figure 3.4.

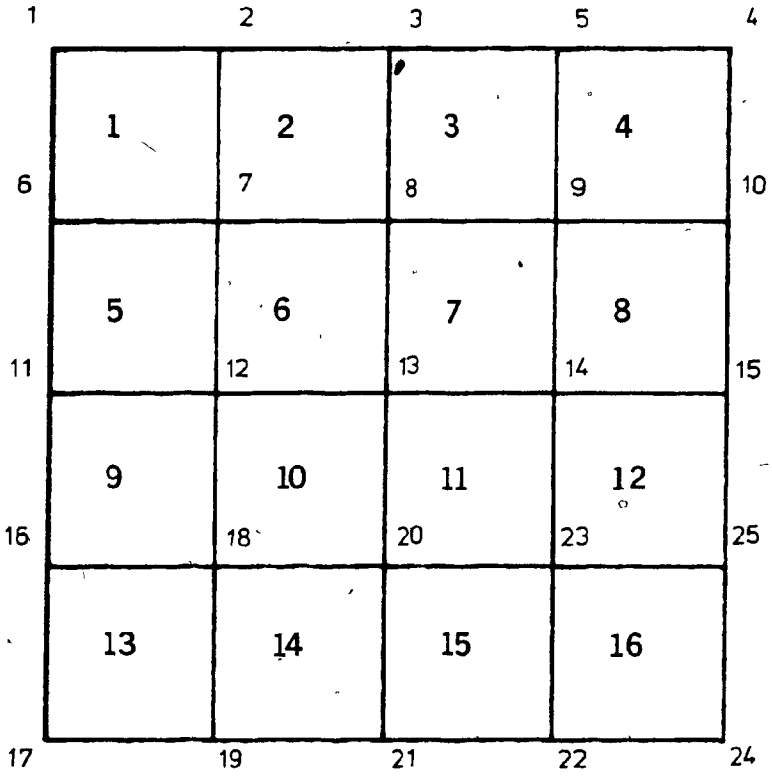


Fig 3.4 An element and node numbering for the mesh in figure 2.1 obtained by a combination of Liu's renumbering strategies.

3.4 Comments on Renumbering for Frontal Solutions

The frontal solution procedure has been developed to allow solution of large sets of finite element equations on small computers. To be consistent, renumbering strategies for frontal solutions should draw on similar frontal lines.

In section 3.2, two direct and two indirect element renumbering strategies were presented. Each direct strategy follows a different procedure for generating an element

sequence that would yield a reduced frontwidth. Akin and Pardue use an element level structure concept, while Liu employes a minimum edge frontwidth growth criterion. The two indirect strategies, on the other hand, are quite similar: both are designed so that the frontwidth does not exceed the minimized bandwidth.

Some generated levels in the Akin-Pardue direct strategy, however, may cause fairly large increases in the size of the active set A. In addition, Liu's strategy possesses an important limitation: it may only be applied on meshes having a constant number of nodes per element. Nevertheless, the computer implementation of these two strategies utilizes adjacency lists whose memory demands correspond to the full element and node set in the finite element mesh.

Indirect element renumbering strategies for frontal solutions clearly do not enjoy the frontal spirit. Firstly, they neglect the eminence of elements over nodes in the frontal procedure; and secondly, because of the storage requirements of node renumbering algorithms, which are proportional to node count [15]. Furthermore, since the node level structure [16] is a basic construct in the Cuthill-McKee (and many other) bandwidth minimizing algorithms, induced element numberings are produced on a level-by-level basis. Very little can therefore be said to favour indirect element renumbering strategies.

Although the application of the simple element-by-element straight node renumbering strategy on a mesh with a level-based element numbering gives a good node numbering in the Cuthill-McKee sense, Liu's node renumbering strategy fulfils the basic frontal requirement that, for a given element sequence, nodes are renumbered in increasing order of their longevities. Numerical stability in the matrix factorization, however, may become the crucial factor in node renumbering [13]. In still another case, some factorization techniques may require a special node numbering; for instance, a low-fill node numbering is needed for better incomplete Choleski decompositions.

In frontal solutions, situations may arise where it is preferable to process sets of elements, or superelements. A choice of the levels in the element level structure as superelements adds another measure of flexibility: processing one such superelement eliminates from the front all the variables associated with the previous one. In this case, destinations are easily allotted by assigning to all the variables (or elements) in a particular superelement its number (plus one). Success in this approach clearly depends on having few elements in each level.

It is well known that increasing the number of levels (the depth) of the level structure decreases the average number of elements in each level, and tends to reduce the maximum number of elements per level (the width) as well.

Such a small width level structure will also overcome the difficulties encountered with the Akin-Pardue direct strategy. Indeed, it is a convenient vehicle in renumbering for frontal solutions, and perfectly fits our criterion. In the next chapter, we present a renumbering strategy based on constructing an element level structure of maximal depth within a frontal framework.

Chapter 4

A Memory-Economic Finite Element and Node Renumbering Strategy for Frontal Solutions

In this chapter, a memory-economic finite element and node renumbering strategy for frontal solutions is introduced. The new strategy is shown to require storage amounting to the maximum size of the active element set A in the frontal procedure, and $O((m*N)^2)$ comparisons, where m is the maximum number of nodes per element and N is the number of elements in the mesh.

4.1 A Memory-Economic Renumbering Strategy

The new memory-economic finite element renumbering strategy generates an element sequence for frontal solutions by constructing an element level structure of maximal depth. Although the procedure followed to obtain such a level structure is similar to that used for its nodal counterpart [13], their implementations differ substantially because of the frontal approach of the new implementation. A detailed description of the new strategy is given below.

I Constructing an element level structure:

Let all the elements {e} in a finite element mesh initially belong to the element set E, and let the final set F be empty. The work begins by choosing an (arbitrary)

element from E to constitute the active set A. At each step of the process, a set B is constructed, comprising all the elements in E which directly adjoin any element in set A. In other words, B is the adjacent set of A fully contained in E, and therefore forms a new level in an element level structure. In the sequel, the two synonymous words set and level may be interchanged. When B has been found, F is enlarged to contain A; all members of B are removed from E; and set B is substituted for set A. This process is continued until no members remain in set E. More formally, the following steps describe the procedure:

1. E := {all elements}
   F := {empty}
   A := {one element in E}

2. E := E - A
   B := {empty}

3. for every e in E
   if Inc(e,A) > 0 then begin B := B + e
   E := E - e end

4. F := F + A
   A := B

5. if |A| > 0 then go to 2

Here Inc(e,A) denotes the number of elements in the set A to which element e is connected; |A| is the number of elements

in A.

In the above description, if the set augmentation in step 4 is done so as to keep F ordered, there will be no need to manipulate F in any way, and the set is very naturally accumulated in a sequential file. On the other hand, step 3 requires examination of every element in E, so that if the set E is kept in a file, the file must be re-read at every pass. The set exclusion (removal of elements from E) also requires set manipulation. These are most easily taken care of by keeping both E and F in sequential files, and by creating an additional, temporary, file G. As E is read, it is partitioned into B and G. After each pass of the algorithm, G is then substituted for E. In practice this substitution is best achieved by reassigning file names, without any need for copying one file into another. In detail, the algorithm then reads:

1. E := {all elements}
   F := {empty}
   A := {one element in E}
   G := E - A

2. E := G
   G := {empty}
   B := {empty}

3. for every e in E
   if Inc(e,A) > 0 then B := B + e

else G := G + e

4. (augment every element in A)

5. F := F + A

   A := B

6. if |A| > 0 then go to 2

As pointed out in chapter 2, much storage will be saved while performing the frontal matrix assembly and factorization with a proper destination allotment. This is carried out in step 4 above for, after step 3, it is an easy matter to determine the highest-numbered element connected to each element in A. The elements in A which are not connected to any element in B and, similarly, those in the last set A may be assigned the highest element number in set A itself. As a final remark, it may be noted that the "frontal" active element set A in chapter 2 is equivalent to the union of the two "computer" sets A and B above.

II Constructing an element level structure of maximal depth:

To construct an element level structure of maximal depth, the work proceeds by choosing an (arbitrary) element from the last element level at which another rooted level structure is constructed in a similar fashion. If the depth of the new level structure is equal to that of the previous one, the process terminates; otherwise, more iterations are performed until this condition has been achieved.

In fact, not all the elements in E need to be examined in step 3 at every pass, during any of the succeeding iterations. Much work can be saved if element destinations are used to mask the components from which each new level is constructed. This can be accomplished simply by assuming that, at each pass, a set M exists, containing all elements in E with destinations less than the lowest element number in the constructed portion of the new level structure. As E is read, the elements which are members of M are written into G immediately so as to keep it ordered for the next passes. Clearly, set M (empty in the first iteration) is never created; rather it is virtually constructed by destination allotment. The algorithm finally reads:

1. d := 1
   L := {all elements}
   E := {all elements}

2. l := 0
   F := {empty}
   A := {one element in L}
   G := E - A

3. l := l + 1
   E := G
   M := {masked elements in E}
   G := {empty}
   B := {empty}

4. for every e in E

   if $e \in M$ then G := G + e else

   if Inc(e,A) > 0 then B := B + e

   else G := G + e

5. (augment every element in A)

6. F := F + A

   A := B

7. if |A| > 0 then go to 3 else

   if l > d then begin d := l

   L := {last set A}

   E := F

   go to 2 - end

Here l gives the depth of the element level structure, whereas set L is never created.

III Element renumbering:

Although the element numbering produced by the algorithm as it stands is good, some minor degree of improvement may be achieved by rearranging the elements in each set so that the difference in sequential numbers of adjacent elements is minimized. This, however, should be done at every iteration since it is not known a priori when the process will terminate. Step 5 then reads:

5. (reorder elements in B)

   (augment every element in A)

This task can be accomplished by generating an array to hold the element number in A to which each element in B is first connected, while performing step 4 above; the elements in B are then "grouped" in a Cuthill-McKee sense. In this case, an examination of every element in A in step 4 is no longer required, which may result in a great saving of work. The element pattern obtained, however, does depend on the initial order of element numbers.

The elements may be renumbered in other convenient ways. For instance, a King-like element numbering can be imposed on the level structure in a way similar to that used by Gibbs [17] in his hybrid profile reduction algorithm. However, this will require generating a "connection table" (or another suitable adjacency structure) [15] for the elements in A (the objects now to be rearranged), and an examination of every element in A in step 4. If a particular element is specified to be renumbered first, only one iteration needs to be performed, and this scheme may be favoured over the less neat (in a heuristic sense) element grouping described above, of course, provided that we can afford the extra storage required. The work involved and the difficulties encountered in updating the destinations would otherwise render this scheme unacceptable.

Other element numbering patterns may still be obtained. The new algorithm lends itself naturally to produce a nested dissection element numbering [12]. First an element level

structure of maximal depth is constructed. The elements in the mid-level that are connected to the next level are then chosen as a separator (ties are broken arbitrarily), and renumbered last. This scheme is repetitively applied to each connected component until all the elements have been renumbered; the elements in any component whose level structure is of depth less than or equal to 1 may be renumbered directly. However, this method of renumbering is liable to require a much more careful component masking and large amount of computer time, and yet still to face similar destination updating problems.

IV On processing levels as superelements:

Processing levels as if they were superelements in this renumbering strategy can lead to great ease in coding. For instance, the elements belonging to each set are readily recognized so that destination allotment is no problem whatsoever. Component masking during the succeeding iterations is also quite simple. Let some elements in level i of the previous level structure be included in the constructed portion of the new one. Then, all elements in the first i-2 levels certainly need not to be examined (members of set M). Furthermore, as will be seen in section 4.3, assessing the number of comparisons performed using superelements is a straight-forward exercise. Such an assessment for individual element processing is difficult, although both should yield the same bound.

V A worked example:

In this section we demonstrate the application of the new algorithm on the finite element mesh in figure 2.1. In the demonstration below, levels-as-superelements are processed; E, F, A, G and B are the element sets described in sections 4.1.I and 4.1.II; the elements in B appear grouped in the way described in section 4.1.III: and the elements discarded by component masking (members of set M) are shown encircled. The augmentation of F by the last set A is omitted below. The first element is arbitrarily chosen to be the starting element; and at each succeeding iteration, the first element in the last level of the previous iteration is taken as the new root. The element numbering obtained in this demonstration is shown in figure 4.1.

The Application of the New Strategy on the Finite Element Mesh in Figure 2.1

The First Iteration

E	F	A	G	B	E	F	A	G	B	E	F	A	G	B	E	F	A	G	B
1		1	2		2		1	2	3	2	1	3		2		1	2		
2			3		3			4	5	4		5		4		3	4		
3			4		4			8	6	8		6		8		5	8		
4			5		5			11	7	11		7		12		6	12		
5			6		6			12	9	12		9		16		7	16		
6			7		7			15	10	15		10		11		9	11		
7			8		8			16	13	16		13		15		10	15		
8			9		9				14			14				13			
9			10		10											14			
10			11		11														
11			12		12														
12			13		13														
13			14		14														
14			15		15														
15			16		16														
16																			

The Second Iteration

E	F	A	G	B	E	F	A	G	B	E	F	A	G	B	E	F	A	G	B	E	F	A	G	B
1		2	1		(1)		2	1	5	1	2	5	6	1	6	2	1		6		2	6			
3			3		3			3	9	3		9	7	14	7	5	14		7		5	7			
5			5		5			6	10	6		10	13	4	13	9	4		13		9	13			
6			6		6			7	8	7		8	11	12	11	10	12		11		10	11			
7			7		7			13	15	13		15		16		8	16				8				
9			9		9			14		14				3		15	3				15				
10			10		10			4		4											1				
13			13		13			12		12											14				
14			14		14			16		16											4				
2			4		4			11		11											12				
4			8		8																16				
8			12		12																3				
12			16		16																				
16			11		11																				
11			15		15																				
15																									

The Third Iteration

E	F	A	G	B	E	F	A	G	B	E	F	A	G	B	E	F	A	G	B	E	F	A	G	B
2		6	2		(2)		6	2	1	(2)	6	1	2	5	2	6	5		2		6	2		
5			5		(5)			5	3	5		3	8	9	8	1	9		8		1	8		
9			9		(9)			9	13	9		13	15	10	15	3	10		4		3	4		
10			10		(10)			10		10			4	14	4	13	14		12		13	12		
8			8		(8)			8		8			12	7	12		7		16		5	16		
15			15		(15)			15		15			16		16				15		9	15		
1			1		1			14		14			11		11				11		10	11		
14			14		14			4		4											14			
4			4		4			12		12											7			
12			12		12			16		16														
16			16		16			7		7														
3			3		3			11		11														
6			7		7																			
7			13		13																			
13			11		11																			
p1																								

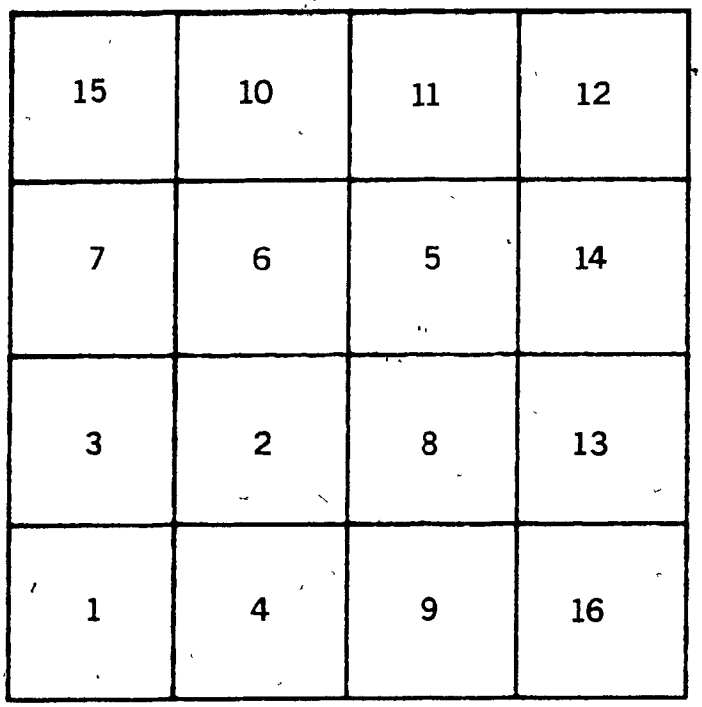


Fig 4.1 The element numbering obtained in the above demonstration for the mesh in figure 2.1.

4.2 Node Renumbering in the New Strategy

Finite elements are commonly represented as sets of node numbers, which in turn serve as pointers to coordinate arrays. For example, a first order triangular element would ordinarily be represented as three integers {i,j,k}, to indicate that its first node was located at x(i), y(i), z(i), and so on. Node renumbering thus consists of two separable tasks: the generation of a renumbering key, and the rearrangement of the arrays of node coordinates.

A renumbering key array K is generated as follows: K(i) is the old node number corresponding to the new node number i. Since the key array is generated in running sequence of the index i, it can be written to file immediately. Care should be taken, however, if constrained nodes are to be renumbered last; or if binary constrained nodes are included. If a particular element is specified to be renumbered first, nodes may be renumbered concurrently with elements in a very simple fashion. Step 5 then reads:

5. (reorder elements in B or A)
   (augment every element in A)
   (record new node numbers)

In this case, it suffices to keep in memory a large enough portion of K to allow the element node numbers to be rewritten, i.e., a portion that covers all elements in sets A and B above. On the other hand, if the algorithm is to iterate, a separate program run is necessary for generating the renumbering key array and rewriting element node numbers. In any event, an element-by-element straight node renumbering is a convenient method to be used in conjunction with any of the element renumbering schemes described in section 4.1.III.

After the key array has been generated and filed, the coordinate arrays usually need to be rearranged. This task is best accomplished in another separate program run that must read as much as possible of the renumbering key array,

K; read the entire coordinate file so as to extract the relevant coordinates; and then write these to an output file. This process is then repeated as many times as necessary. It may be seen that the structure of this task very closely parallels that of element renumbering, so that it is unnecessary to give detailed algorithms.

4.3 Memory and Computer Time Limitations

In this section, we consider the (real) memory and computer time limitations for the basic algorithm as presented in sections 4.1.I and 4.1.II; that is, for constructing an element level structure of maximal depth.

The storage requirements of this algorithm are immediately seen. The detailed description given above clearly indicates that the sets of memory-resident elements at the different stages of the renumbering procedure are exactly the same as the active element sets in the frontal procedure. It is hard to give any bounding values for the sizes of these sets since they largely depend on the mesh topology and the choice of the starting element.

The estimation of the number of comparisons performed, however, needs a little algebra. Let m be the maximum number of nodes per element, and let N be the number of elements in the finite element mesh. The number of comparisons performed in the first iteration ($N.C_1.$) is certainly

$$N.C_1. \leq m^2*\{(N-1) + w_1*(N-1-w_1) + w_2*(N-1-w_1-w_2) + \ldots + w_{\ell-1}*(N-1-\ldots-w_{\ell-1})\}, \quad (4.1)$$

$$= m^2*\{N*(1+w_1+w_2+\ldots+w_\ell) - (1 + w_1*(1+w_1) + \ldots + w_\ell*N)\}, \quad (4.2)$$

where w_i is the number of elements in level i, and l is the number of levels. Since $w_i \geq 1$, the number of comparisons becomes

$$N.C_1. < m^2*\{N^2 - N*(N+1)/2\} = m^2*N*(N-1)/2 \quad (4.3)$$

This represents the main fraction of work; that due to subsequent iterations has been significantly slashed with the proper masking of element components. To illustrate this point, consider the case of processing levels-as-superelements; the number of comparisons performed in the second iteration ($N.C_2.$) is then

$$N.C_2. \leq m^2*\{(w_\ell+w_{\ell-1}-1) + v_1*(w_\ell+w_{\ell-1}+w_{\ell-2}-1-v_1) + \ldots + v_{d-1}*(N-1-v_1-\ldots-v_{d-1})\}, \quad (4.4)$$

$$= m^2*\{(v_1+r_1) + v_1*(v_2+r_2) + \ldots + v_{d-1}*v_d\}, \quad (4.5)$$

where v_i is the number of elements in level i, and d is the number of levels in the second level structure. Although it is hard to assess the remainders r_i, they can be generally considered of O(v), where $v = \max\{v_i\}$. It immediately follows that $N.C_2.$ is of $O(m^2*v^2*d)$. Since v*d is of O(N), this is obviously of $O(m^2*t*N)$ with $t \ll N$. The number of comparisons in other iterations will be further reduced as

level structures of smaller widths are generated. Summing up, the total number of comparisons will be

$$m^2 * \{N^2/2 + \sum_{i=1}^{p} t_i * N\}, \qquad (4.6)$$

where p is the number of iterations, and t_1 is -0.5 (equation 4.3).

Comparison counts appear not to be a reliable measure of the actual cost of this algorithm. It is generally accepted that memory storage and execution time cannot be reduced simultaneously; an improvement of one normally leads to a deterioration of the other. Since memory has been our primary concern, the new algorithm was developed so as to reduce storage requirements, certainly at the expense of execution time. A substantial overhead due to many file retrievals and memory-disk transfers is therefore expected.

4.4 Two Illustrative Model Problems

The limitations of this renumbering algorithm may be further illustrated by considering two model problems, one two-dimensional and one three-dimensional. In two dimensions, consider a square array of N quadrilateral elements (N a perfect square), with the starting element chosen at one corner of the square at each iteration, so that only two iterations need to be performed.

It will be clear from figure 2.1, or any larger pattern of similar type, that the maximum difference between

adjoining element numbers will be 2 sqrt(N) - 1, and that the number of execution passes per iteration is sqrt(N). The largest number of elements in set A is approximately 2 sqrt(N), so that the memory-resident number of elements is approximately 4 sqrt(N). If memory is available to house K elements concurrently, a problem involving approximately $(K/4)^2$ elements can be renumbered, in 2*(K/4) passes. To illustrate numerically, suppose K = 1000, which represents a very modest-sized minicomputer. Renumbering the model problem takes approximately 500 passes in this case. The renumbering capacity amounts to approximately 62500 elements, equivalent to 250000 nodes if nine-noded quadrilaterals are employed. The number of comparisons performed is $O(10^{11})$.

Consider next a similar three-dimensional model problem: a regular cubic lattice of N brick elements (N a perfect cube). In this case, the number of memory-resident elements is $6*N^{(2/3)}$, and the number of required passes per iteration is approximately $N^{(1/3)}$. If K elements can be housed in memory, $(K/6)^{(3/2)}$ total elements can be dealt with, in 2 sqrt(K/6) passes. Taking K = 1000 as an example, 26 passes of computation will deal with about 2100 elements, or about 18000 nodes if 27-noded brick elements are used. The comparisons count up to $O(10^9)$.

It may be observed in passing that the ratio of required passes for the two-dimensional model to that for

the three-dimensional model is approximately 20. This reflects the fact that the ratio of the elements in the two models is near this figure (approximately 30). A definite reduction in overhead can be achieved by using fewer elements.

Chapter 5

Experimental Results

The evaluation of renumbering algorithms is usually carried out on a computer through extensive experimental tests. In this chapter, we test the new algorithm on a set of meshes that range from relatively small sizes, up to approximately 1000 elements, and that have different topological structures. Some comments will then be drawn on its performance from the results obtained.

5.1 Data Structure -- Element File Organization

In our computational implementation, element files are organized as sequential files of unformatted (binary) records of 32 words each, with one element per record. For each record, the words contain the element destination D, an element type indicator T, a region label R (which serves to identify material properties and source densities), the number of nodes in the element N and a list of node numbers n(i). The type T and the region label R are taken as two-character alphabetics, whereas the numerics are all integers. An element may have up to 28 nodes under this arrangement. Figure 5.1 shows the structure of a typical record in element files.

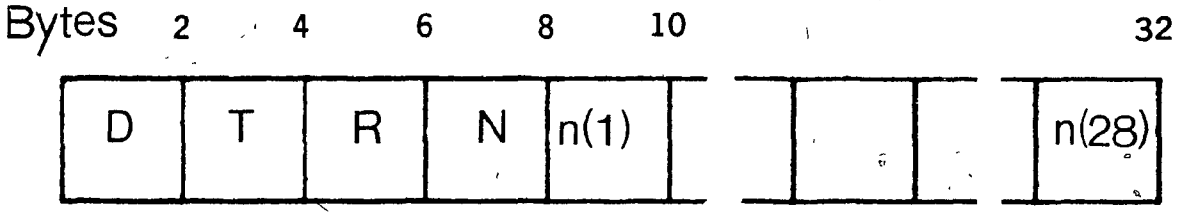


Fig 5.1 Record structure in element files.

5.2 Experimental Results

Existing renumbering algorithms are largely heuristic. Since there are no strict theoretical grounds to evaluate these algorithms, this is done empirically on a computer through extensive experimental tests. These tests need to be performed on a set of problems that arise in practical finite element applications, and that range up to reasonably large sizes, with different topological structures [8].

The set of test problems employed contains five mesh

problems especially designed for testing renumbering algorithms: a rectangular domain, a rectangular domain with a hole, an L-shaped domain, an H-shaped domain and a +-shaped domain. These meshes were produced by MagMesh (the geometric modeller phase of the MagNet 11 field analysis system [22]) through mirror-and-join operations. Since MagMesh is intended to provide only first order triangular elements, these meshes have more elements than nodes (about twice as many elements as there are nodes for large models) [9]. For each mesh, we have considered four different cases consisting of relatively few elements, up to approximately 1000 in number. Representative test problems are shown in figures 5.2-5.6. In addition, three other problems (shown in figures 5.7-5.9) were used in our experiments. The Half-Pole Pitch problem (courtesy of the General Electric Company (GEC), UK) is typical of the problems that arise in machine design. The Bullet problem was generated by a Movie.Byu facility [24]. The third problem (mesh 23) is a geometric display formed by MagMesh. Finally, the Circle problem is taken from reference [25]. Test problem statistics are presented in table 5.1.

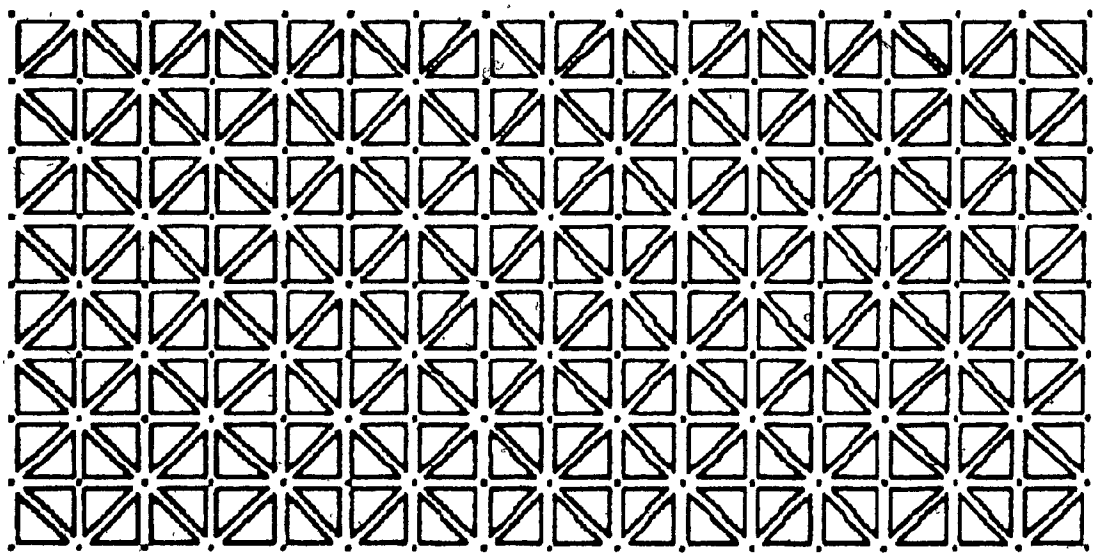


Fig 5.2 A 256 elements, 153 nodes Rectangular Domain (mesh 2).

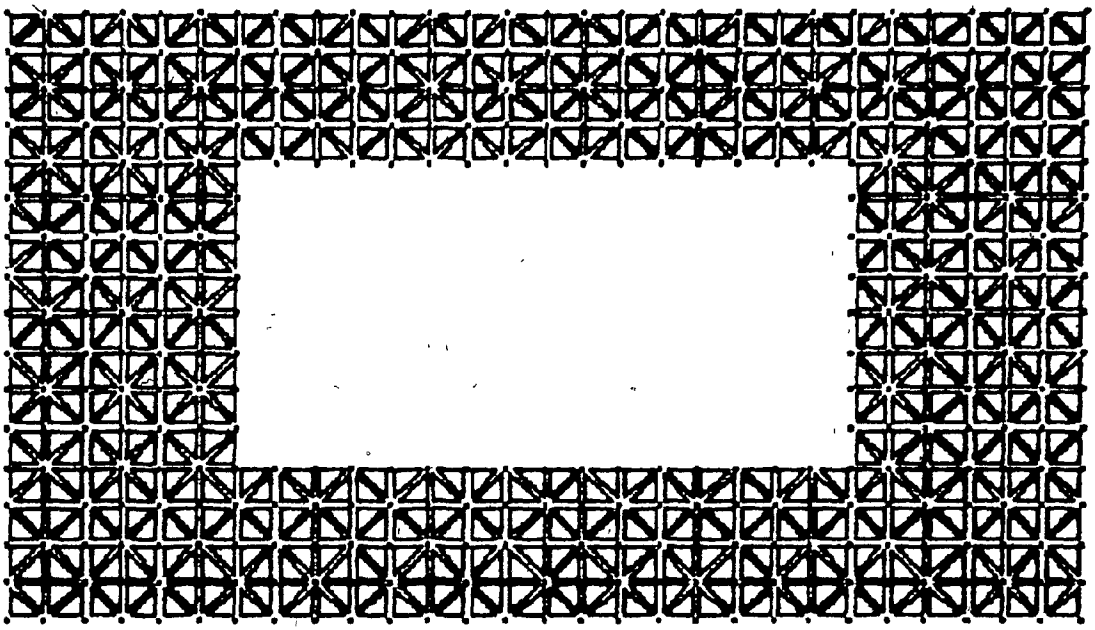


Fig 5.3 A 640 elements, 388 nodes Rectangular Domain with a Hole (mesh 7).

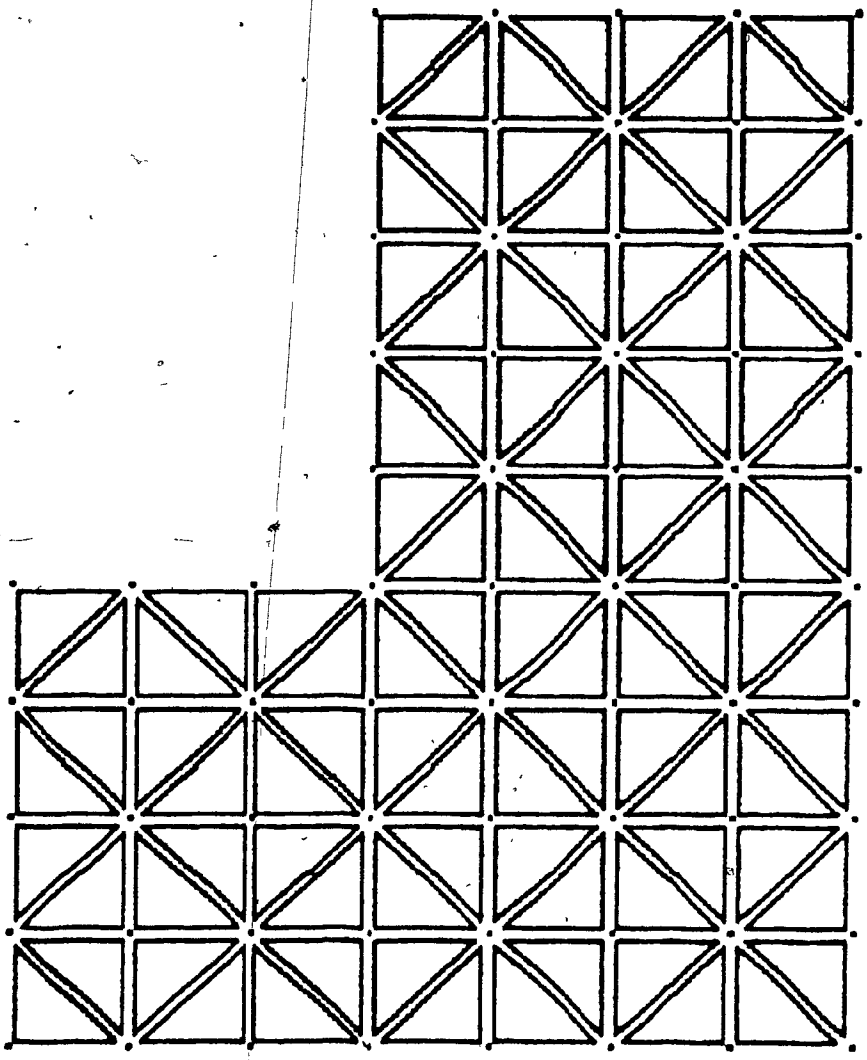


Fig 5.4 A 96 elements, 65 nodes L-shaped Domain (mesh 9).

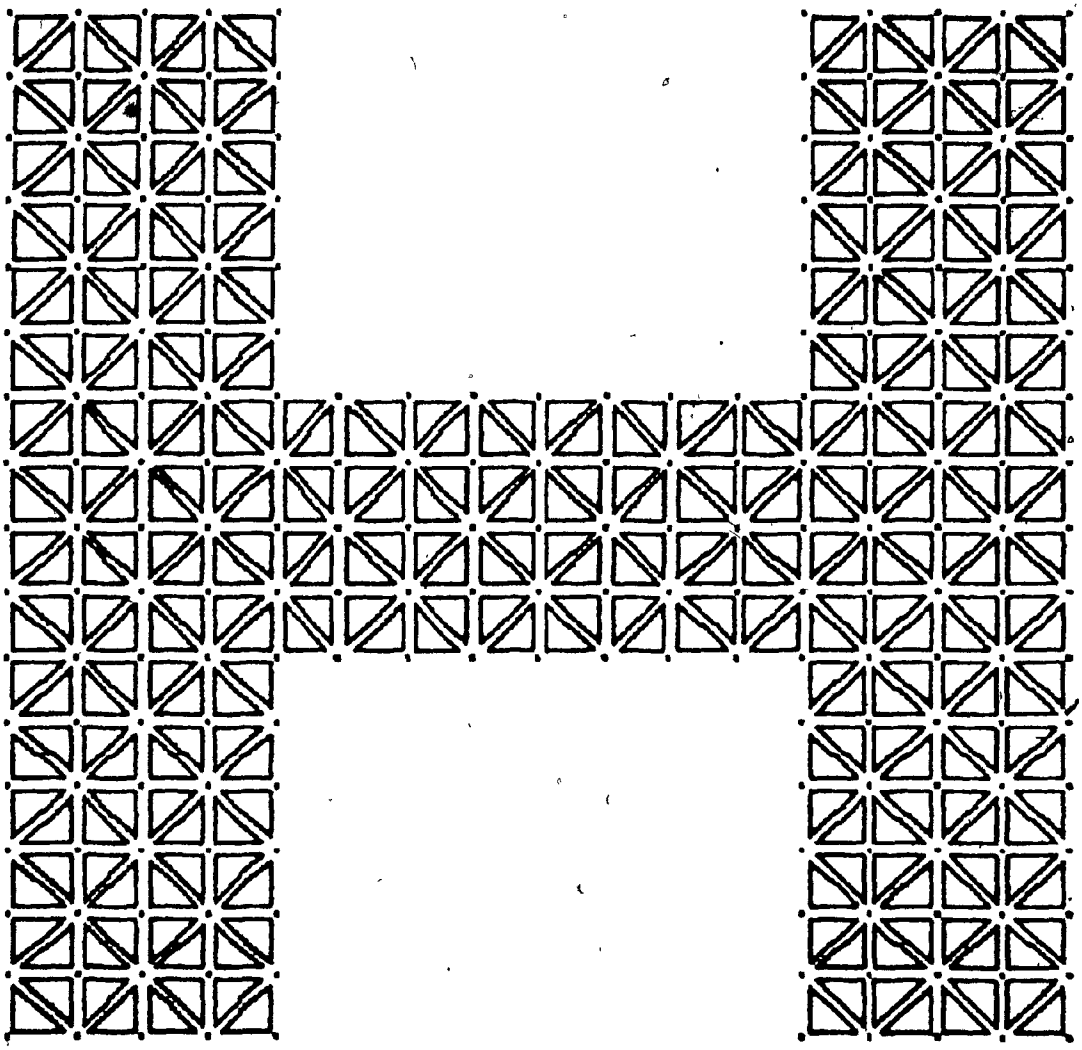


Fig 5.5 A 320 elements, 205 nodes H-shaped Domain (mesh 14).

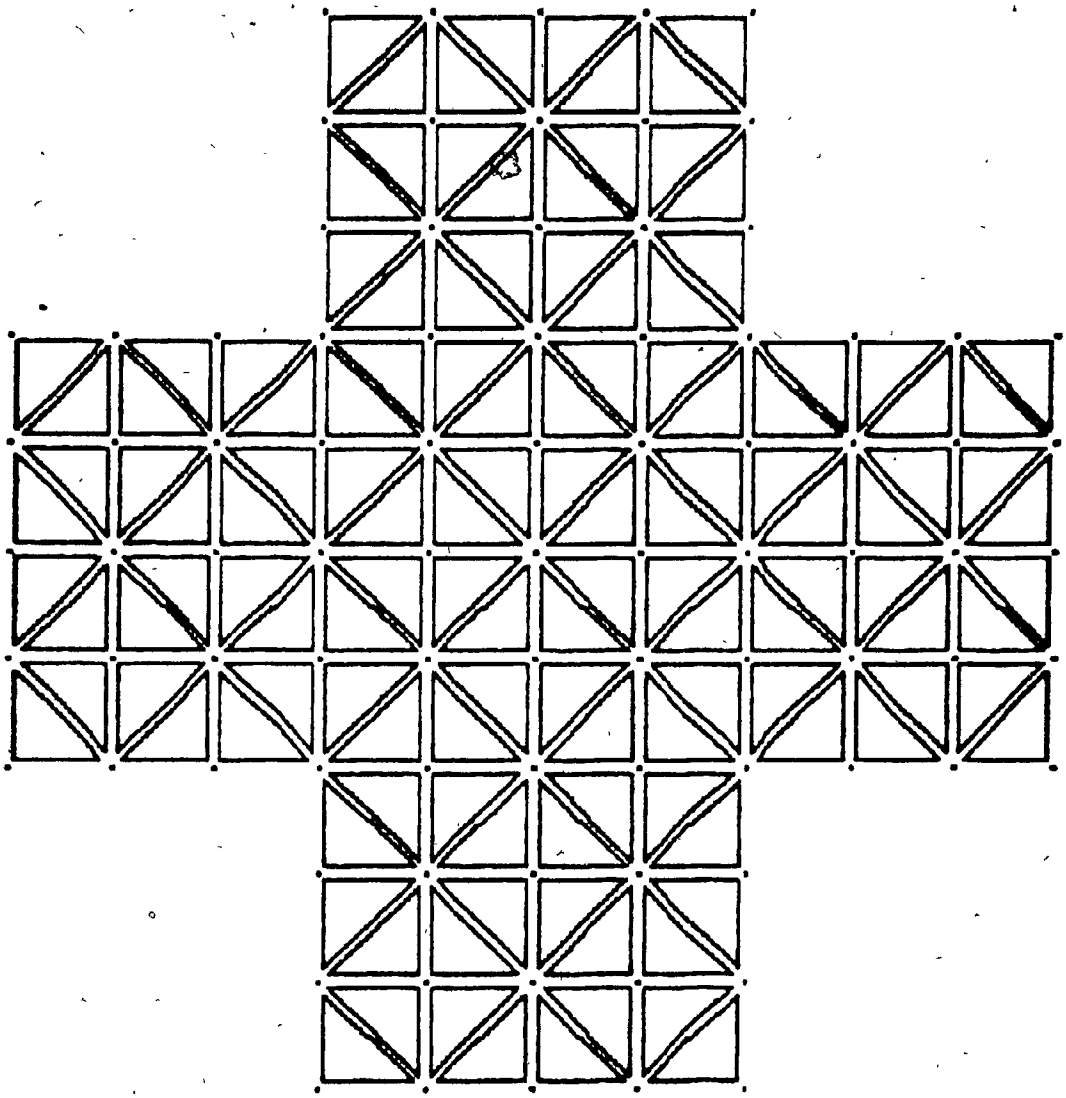


Fig 5.6 A 128 elements, 85 nodes +-shaped Domain (mesh 17).

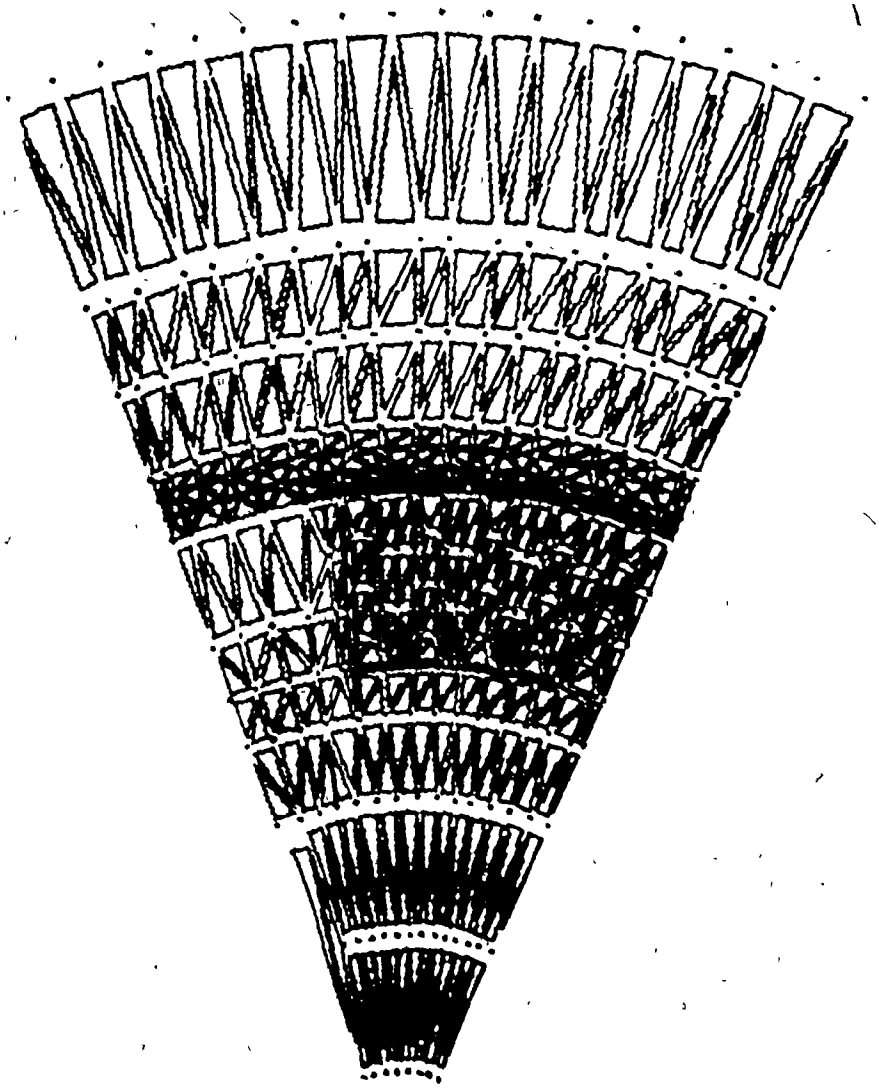


Fig 5.7 A 592 elements, 322 nodes Half-Pole Pitch (mesh 21).

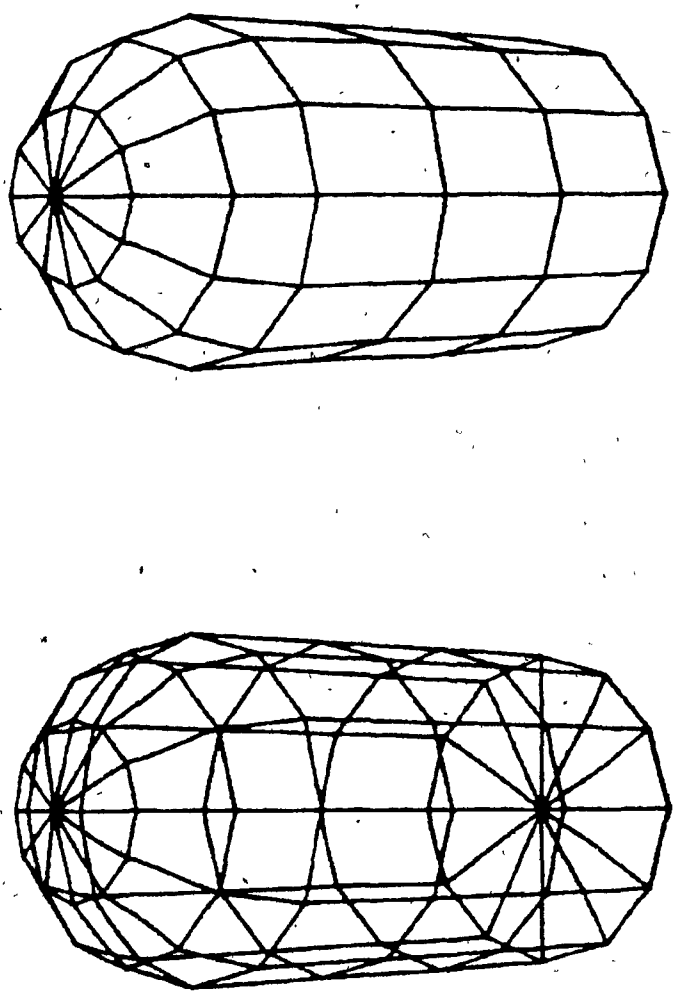


Fig 5.8 A 84 elements, 74 nodes Bullet (mesh 22).

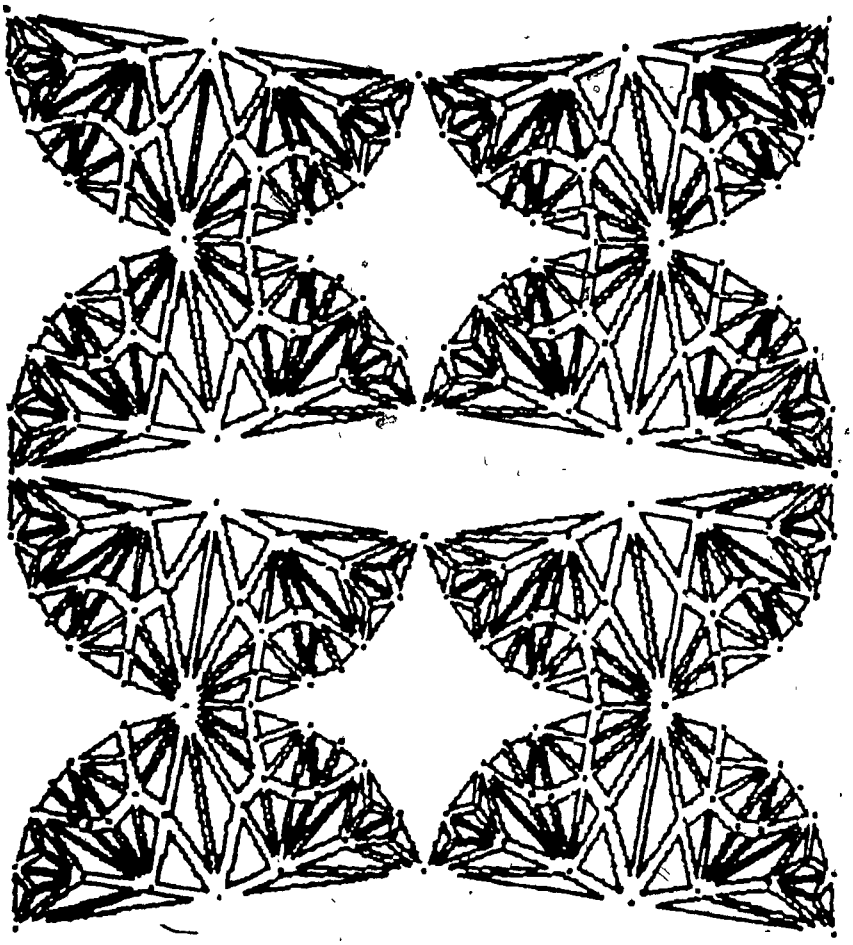


Fig 5.9 A geometric display of 384 elements and 242 nodes (mesh 23).

Table 5.1 Test Problem Statistics

	Mesh	|FE|	|X|	x/e	PR	BW	FW	RMS WF
1	Rectangular Domain	128	81	3	871	58	18	11.4789
2		256	153	3	2162	80	21	14.7887
3		512	289	3	6116	151	32	22.1706
4		1024	561	3	19486	295	57	36.7482
5	Rectangular Domain with a Hole	152	106	3	1136	70	18	11.2862
6		416	254	3	4264	132	27	17.5528
7		640	388	3	7636	199	34	20.6826
8		1080	648	3	13132	330	35	[illegible].1647
9	L - shaped Domain	96	65	3	574	42	13	9.2769
10		350	208	3	2941	73	21	14.6981
11		600	341	3	4929	148	22	14.9928
12		960	539	3	18363	282	51	35.6605
13	H - shaped Domain	152	103	3	1261	75	21	13.0514
14		320	205	3	3326	149	29	17.5799
15		576	343	3	7490	180	38	23.1245
16		1056	608	3	18033	348	48	31.4813
17	+ - shaped Domain	128	85	3	1079	60	19	13.6136
18		240	147	3	2423	108	27	17.3725
19		512	297	3	7582	216	44	26.8480
20		976	545	3	22059	422	67	42.3742
21	Half - Pole Pitch	592	322	3	10464	319	53	34.1798
22	Bullet	84	74	4,3	1270	63	24	18.2409
23		384	242	3	5460	135	36	23.7459
24	Circle	24	73	8,6	1295	39	27	19.0209

Here |FE| denotes the number of elements in the mesh, |X| the number of nodes, x/e the number of nodes per element, PR the profile, BW the bandwidth, FW the frontwidth and RMS WF the root-mean-square wavefront.

The new algorithm was implemented in FORTRAN, and all the tests were run on the McGill University, Computational Analysis and Design Laboratory LSI-11 single user minicomputer, version 3B with EIS/FIS option. In our program, levels-as-superelements are processed; linear insertion [15] is used to group the elements as described in section 4.1.III; and an element-by-element straight node renumbering is used.

The experimental results are presented in tables 5.2-5.7. In these tables, operations mean comparisons other than with zeros; time is in seconds; the number of operations and time of node renumbering are those taken to generate the key array; the depths given read plus one more than is formally defined; and the matrix parameters in the last four columns are as defined in appendix A. It should be noted that we report here, the frontwidth of the matrix, which is in general different from that of the frontal procedure defined in chapter 2. This is motivated by a particular application of the new algorithm [23]. The four rows of results for each mesh correspond to four different starting elements: $1 + [i*N/4]$ with $0 \leq i \leq 3$, where $[i*N/4]$ denotes the largest integer less than or equal to $i*N/4$, N being the number of elements in the mesh. The average results are presented in table 5.8 where the operation counts only are rounded to the next larger integer.

Mesh	Iter	Element Numbering		Depth	Width	Node Numbering		Profile	Band-width	Front-width	rms wave front
		Operations	Time			Operations	Time				
1	2	79098	152	8	26	4602	12	993	17	17	12.9066
	3	114693	216	8	26	4666	12	1015	20	20	13.2642
	3	124952	233	8	26	4653	13	953	18	18	12.4201
	3	116963	219	8	26	4617	13	997	17	17	12.9238
2	3	394714	609	16	26	8280	19	1676	17	17	11.3287
	3	328598	544	16	26	8285	19	1698	18	18	11.5050
	3	327585	547	16	26	8278	18	1659	18	18	11.2317
	3	325226	548	16	26	8299	19	1699	17	17	11.4849
3	3	2375896	3224	17	62	35845	62	6500	34	34	23.8863
	3	1893541	2587	16	52	33301	58	6202	30	30	22.2619
	3	1812103	2500	16	58	34097	58	6321	36	36	23.0687
	2	1359883	1844	16	52	33223	58	6099	30	30	21.0687
4	3	6136574	8724	32	52	61590	104	11025	30	30	20.1489
	3	7408515	10360	32	52	61456	105	10961	30	30	20.0454
	3	6933444	9768	32	52	61443	105	10964	30	30	20.0349
	3	8399796	11630	32	52	61535	104	10978	30	30	20.0748

Table 5.2: Rectangular Domain

Mesh	Iter	Element Numbering		Depth	Width	Node Numbering		Profile	Band-	Front-	rms
		Operations	Time			Operations	Time		width	width	wave front
5	2	107251	213	12	20	4804	13	1099	18	18	10.8589
	2	101863	214	14	22	4419	12	1013	16	16	10.1074
	3	136250	280	12	20	4804	13	1093	18	18	10.8084
	2	101368	213	14	22	4380	13	1019	16	16	10.1892
6	3	1021251	1549	21	42	19351	36	3852	26	26	16.1179
	3	952400	1508	23	50	19424	36	3895	30	30	16.9491
	3	971014	1509	21	42	19413	36	3855	26	26	16.0987
	3	956035	1504	22	46	19582	37	3848	28	28	16.4012
7	3	2191402	4117	31	44	29449	64	5767	26	26	15.6084
	2	2129925	3634	25	56	39744	81	7918	36	36	21.9759
	2	2107306	3591	25	52	36714	76	7303	34	34	19.8529
	2	2129083	3589	25	56	39738	80	7717	33	33	21.1441
8	2	5432494	7981	45	38	53974	93	10528	24	24	16.7604
	2	5472590	8030	45	38	54441	94	10651	25	25	16.9755
	2	5472200	8020	45	38	54405	93	10678	25	25	17.0239
	2	5694937	9056	49	38	49381	87	9598	24	24	15.2999

Table 5.3: Rectangular Domain with a Hole

Mesh	Iter	Element Numbering		Depth	Width	Node Numbering		Profile	Band-	Front-	rms
		Operations	Time			Operations	Time		width	width	wave front
9	2	43885	96	8	22	2561	8	603	15	15	9.8832
	3	67397	140	8	22	2533	8	572	13	13	9.2354
	3	58665	126	8	22	3110	9	688	18	18	11.6758
	2	48331	103	8	22	2520	9	559	13	13	9.0358
10	3	956872	1670	18	40	14958	34	2980	24	24	15.2116
	3	926959	1660	18	40	14954	35	2977	24	24	15.2045
	3	873886	1552	18	40	14951	34	2979	23	23	15.1998
	3	952941	1666	18	40	14936	35	2970	23	23	15.1762
11	3	1998493	3225	40	20	19274	38	3458	12	12	10.4199
	3	1868121	3026	40	20	19330	39	3498	13	12	10.5486
	3	2314493	3539	40	20	19383	39	3494	13	12	10.5358
	3	2078526	3253	40	20	18711	38	3471	13	12	10.4642
12	3	6050942	10702	38	58	51739	108	9506	32	32	19.0165
	3	5832589	10404	38	58	51855	108	9568	32	32	19.1142
	3	5572488	10059	38	58	51011	107	9474	32	32	18.9104
	3	5778859	10221	38	58	51909	108	9528	32	32	19.0516

Table 5.4: L-Shaped Domain

Mesh	Iter	Element Numbering Operations	Element Numbering Time	Depth	Width	Node Numbering Operations	Node Numbering Time	Profile	Band-width	Front-width	rms wave front
13	2	126378	235	10	28	5120	14	1174	18	18	12.0040
	3	171956	319	10	28	5136	14	1213	18	18	12.3355
	3	180603	331	10	28	5134	13	1190	18	18	12.1256
	3	197577	366	10	34	5680	15	1223	22	22	12.9168
14	3	825592	1525	20	36	12252	29	2605	23	23	13.6498
	3	877359	1569	20	36	12252	29	2605	23	23	13.6498
	3	803404	1470	20	36	12259	29	2603	23	23	13.6330
	3	821039	1500	20	36	12246	29	2609	23	23	13.6627
15	2	2147564	3628	22	60	36874	76	7044	38	38	22.5043
	3	3071337	5182	22	60	36673	76	7004	39	38	22.4286
	3	2846025	4871	22	60	36889	76	7064	38	38	22.5665
	3	3049476	5156	22	60	36973	76	7064	39	38	22.5527
16	3	9869240	16842	33	78	82423	160	14742	46	46	26.2138
	3	10744456	18046	33	78	82154	160	14979	44	44	26.6432
	3	10541097	17699	33	78	82192	160	14990	45	45	26.5381
	3	10466413	17597	33	78	82236	161	14982	44	44	26.5536

Table 5.5: H-Shaped Domain

Mesh	Iter	Element Numbering		Depth	Width	Node Numbering		Profile	Band-	Front-	rms
		Operations	Time			Operations	Time		width	width	wave front
17	2	77552	159	10	26	4388	12	935	18	18	11.8793
	3	110871	218	10	26	4419	12	955	18	18	12.1641
	3	101810	210	10	26	4434	12	951	18	18	12.0640
	3	102740	207	10	28	4299	11	934	19	19	11.8620
18	3	371186	663	12	32	11032	26	2225	23	23	15.8901
	3	390849	700	14	34	10364	25	2045	22	22	14.8734
	2	263667	480	14	34	10313	25	2015	22	22	14.6052
	3	356117	653	14	36	9886	24	1973	23	23	14.3178
19	3	1450303	2621	24	44	23516	52	4464	27	27	15.9591
	3	1831116	3167	24	44	23615	52	4494	27	27	16.0536
	3	1833601	3170	24	44	23673	52	4505	27	27	16.1131
	2	1150031	2003	24	44	23523	52	4472	27	27	15.9810
20	3	6753462	11259	24	98	94385	182	16657	54	54	33.3137
	3	4840530	8575	32	72	67671	136	12237	41	41	24.3369
	3	5148000	8971	32	72	67247	135	12005	41	41	24.0222
	3	6082820	10432	32	72	68373	136	12115	40	40	24.1463

Table 5.6: +-Shaped Domain

Mesh	Iter	Element Numbering Operations	Element Numbering Time	Depth	Width	Node Numbering Operations	Node Numbering Time	Profile	Band-width	Front-width	rms wave front
21	2	1715703	2407	22	44	33991	60	5947	27	27	19.2953
	3	2481434	3517	23	43	34648	60	5936	26	26	19.2467
	4	2965996	4284	23	43	34709	61	5934	25	25	19.2002
	3	2511862	3487	21	44	34330	60	6074	27	27	19.5456
22	2	45724	77	7	18	4917	11	1170	23	22	16.9674
	2	44407	76	7	22	4927	11	1111	23	23	15.9251
	2	47032	79	7	14	4213	9	943	17	16	13.2517
	2	42642	74	7	22	4902	10	1100	23	23	16.0876
23	3	1176372	1925	13	52	32223	63	6910	46	46	30.4116
	2	779132	1284	14	55	32574	63	7186	49	49	32.0685
	3	1076020	1815	14	55	32578	63	6939	48	48	31.0071
	3	1065769	1790	14	55	32760	63	7198	49	49	32.0642
24	3	15742	40	5	9	3552	8	1414	29	29	20.8642
	2	9826	27	5	9	3559	8	1416	31	31	20.8664
	2	9293	26	5	9	3554	8	1414	29	29	20.8642
	2	8973	25	4	11	4288	9	1623	43	39	24.5265

Table 5.7: Meshes 21-24

Mesh	Iter	Element Numbering		Depth	Width	Node Numbering		Profile	Band-	Front-	rms
		Operations	Time			Operations	Time		width	width	wave front
1	2.75	91129	205	8	26	4635	12.5	989.5	18	18	12.6287
2	3	344031	562	16	26	8286	18.75	1683	17.5	17.5	11.3876
3	2.75	1860356	2539	16.25	56	34117	59	5280.5	32.5	32.5	22.3214
4	3	7469583	10120.5	32	52	61506	104.5	10982	30	30	20.0760
5	2.25	111683	230	13	21	4602	12.75	1056	17	17	10.4910
6	3	975175	1517.25	21.75	45	19443	36.25	3462.5	27.5	27.5	16.3917
7	2.25	2139430	3732.75	26.5	52	36412	75.25	7176.25	32.25	32.25	19.6453
8	2	5518056	8271.75	46	38	55551	91.75	10363.75	24.5	24.5	16.5149
9	2.5	54570	116.25	8	22	2681	8.5	605.5	14.75	14.75	9.9326
10	3	927665	1637	18	40	14950	34.5	2976.5	23.5	23.5	15.1980
11	3	2064909	3261	40	20	19175	38.5	3480.25	12.75	12	10.4921
12	3	5808720	10346.5	38	58	51629	107.75	9519	32	32	19.0232

Table 5.8: Average Results

Mesh	Iter	Element Numbering		Depth	Width	Node Numbering		Profile	Band-	Front-	rms
		Operations	Time			Operations	Time		width	width	wave front
13	2.75	169129	312.75	10	29.5	5268	14	1200	19	19	12.3455
14	3	831849	1516	20	36	12253	29	2605.5	23	23	13.6489
15	2.75	2778601	4709.25	22	60	36853	76	7044	38.5	38	22.5130
16	3	10405302	17546	33	78	82252	160.25	14923.25	44.75	44.75	26.4872
17	2.75	98244	198.5	10	26.5	4385	11.75	943.75	18.25	18.25	11.9924
18	2.75	345455	624	13.5	34	10399	25	2064.5	22.5	22.5	14.9216
19	2.75	1566263	2741	24	44	23582	52	4483.75	27	27	16.0267
20	3	5706203	9809.25	30	78.5	74419	147.25	13253.5	44	44	26.4548
21	3	2418749	3423.75	22.25	43.5	34420	60.25	5972.75	26.25	26.25	19.3470
22	2	44952	76.5	7	19	4740	10.25	1081	21.5	21	15.5580
23	2.75	1024324	1703.5	13.75	54.25	32534	63	7058.25	48	48	31.3879
24	2.25	10959	29.5	4.75	9.5	3739	8.25	1466.75	33	32	21.7803

5.3 On the Performance of the New Algorithm

Some general comments on the performance of the new algorithm can be drawn from the set of tabulated results.

A first and most immediate observation is that the number of iterations is almost always three; two iterations were sufficient in quite a few runs, while four iterations was required only once in the third run for the Half-Pole Pitch problem (mesh 21). Although it is not a rule, this very few number of iterations required represents an important feature of the algorithm, which can be exploited to reduce the computing cost, for example, by restricting the number of iterations to only two.

Operation counts for element renumbering indicate that the comparison bound for constructing the level structure in section 4.3 is an overestimate; in practice, it is more like $O(m*N^2)$, where m and N are as before, the respective maximum number of nodes per element and the number of elements in the mesh. This bound clearly illustrates the effectiveness of the grouping technique in reducing the overall number of comparisons.

As was expected, comparison counts are not an accurate measure of the actual computer time expended on element renumbering. Certainly, counting the number of comparisons performed requires a similar number of additions to be done. It may also be noted, however, that all element data must be

manipulated as well, which adds to the overhead due to the many file retreivals and memory-disc transfers. The sources of overhead, together with the practical comparison bound above, readily suggest that a substantial reduction in the total cost can be achieved by using few higher order elements instead of many lower order elements, or by a proper manual substructuring of the mesh beforehand (which can be done easily for some finite element meshes: for example, mesh 23).

The widths obtained are either equal or very close to each other in the different runs for most of the problems. A large deviation is found only in the 976 element +-Shaped Domain problem (mesh 20). Although our set of problems contains topologically different structures, these widths, as an examination of the sixth column in the tables may prove, are in general of O(sqrt(N)). The maximum number of memory-resident elements at any stage of the renumbering procedure (usually in the first iteration), however, depends on the mesh topology and the choice of the starting element. It is important to report here that the arrays used in our program are all integers of fixed dimension 150 long.

The number of comparisons performed and the time expended on generating the renumbering key array are negligible compared to those of element renumbering. However, this is not likely to be the case if higher order elements are used so that the number of nodes is much larger

than the number of elements in the mesh. Keeping in mind that the node renumbering phase may be entirely omitted, a choice between doing less work and obtaining more computational efficiency must be made.

In spite of the very simple element and node renumbering techniques used, the algorithm consistently reduced the original bandwidths significantly and, in most of the runs, produced better profiles and frontwidths. The worst results were for mesh 23 which exhibits a special topology. It is possible, however, that the usual reversal of the element and node numberings would produce a better pattern [21]. The King-like scheme in section 4.1.III could be used instead to give a minimum profile element and node numbering, if we can afford the extra work and storage involved. A low-fill numbering can be obtained by using the nested dissection scheme described in the same section, but this is liable to require a large amount of computer time.

To conclude, the experimental testing of the new algorithm clearly indicates that it has a reliable performance and great memory saving ability. Furthermore, the choice of the starting element appears to only have a small impact on its performance. Indeed, the reason the new algorithm works the way it does is directly related to the maximal depth element level structure it generates.

Chapter 6

Conclusion

A new memory-economic finite element renumbering strategy for frontal solutions was developed in this thesis. The new strategy is based on constructing an element level structure of maximal depth within a frontal framework. The memory-resident element sets at the different stages of the renumbering procedure are exactly the same as the active element sets in the frontal procedure. In terms of operation count, $O((m*N)^2)$ comparisons are required for constructing such a level structure, where m and N are the maximum number of nodes per element and the total number of elements in the finite element mesh respectively.

Three different element renumbering schemes were considered: a Cuthill-McKee-like element grouping, a King-like element renumbering and the nested dissection technique. We favoured the simpler element grouping in our experiments because of its advantages in computer space, time, and destination allotment. Together with the element grouping, an element-by-element straight node renumbering was used.

Extensive experimental testing of the new algorithm was carried out on the McGill University, Computational Analysis and Design Laboratory LSI-11 single user minicomputer. The

set of test problems employed consists of 24 meshes ranging in size from the relatively small ones to those containing approximately 1000 elements, with different topological structures. For each mesh, four different starting elements were tried so that we would be able to clearly assess the performance of the new algorithm.

The experimental results clearly indicate that the new algorithm has a reliable performance and great memory saving ability. The choice of the starting element had only a small impact on such a performance. The level widths obtained were either equal or very close to each other in the different runs for most of the problems. In general, they were of O(sqrt(N)). Furthermore, in spite of the simple techniques used for element and node renumbering, the new algorithm consistently reduced the original bandwidths of the test problems significantly and, in most of the runs, produced better profiles and frontwidths.

The algorithm presented in this thesis is an important step in the development of frontal solutions since it permits a "frontal" finite element and node renumbering. It is felt that the element sequences the new algorithm generates are at least as good as those obtained by any other means. The immediate extension of the new algorithm to produce nested dissection element numbering is particularly of great significance. Indeed, the very modest storage requirements of this algorithm and, more important,

its consistency with the frontal procedure endorse it as a practical renumbering scheme for frontal solutions.

References

[1] Abbas, S.F.; "Some Novel Applications of the Frontal Concept"; Inter. J. for Numer. Meth. in Engg; Vol. 15; 519-536 (1980).

[2] Akin, J.E. and Pardue, R.M.; "Element Resequencing for Frontal Solutions"; in The Mathematics of Finite elements and Applications; Whiteman, J.R. (Editor); 535-541; Academic Press; New York; USA (1974).

[3] Bathe, K.J. and Wilson, E.L.; "Solution Methods for Eigenproblem in Structural Mechanics"; Inter. J. for Numer. Meth. in Engg; Vol. 6; 213-226 (1973).

[4] Chari, M.V.K. and Silvester, P.P. (Editors); Finite Elements in Electrical and Magnetic Fields; A Wiley-Interscience Publ. (1980).

[5] Collins, R.J.; "Bandwidth Reduction by Automatic Renumbering"; Inter. J. for Numer. Meth. in Engg; Vol. 6; 345-356 (1973).

[6] Csendes, Z.J., Minhas, F.U. and Silvester, P.P.; "Universal Finite Element Matrices for Tetrahedra"; To be published.

[7] Cuthill, E. and McKee, J.; "Reducing the Bandwidth of Sparse Symmetric Matrices"; Proc. 23rd Nat. Conf. Assoc. Comput. Mach.; ACM Publ.; 157-172 (1969).

[8] Everstine, G.C.; "A Comparison of Three Resequencing Algorithms for the Reduction of Matrix Profile and Wavefront"; Inter J. for Num. Meth. in Engg; 837-853 (1979).

[9] Ewing, D.J.F., Fawkes, A.J. and Griffiths, J.R.; "Rules Governing the Numbers of Nodes and Elements in a Finite Element Mesh"; Inter. J. for Numer. Meth. in Engg; Vol. 2; 597-600 (1970).

[10] George, A.; "Computer Implementation of the Finite Element Method"; Ph.D. Thesis; Computer Science Dept.; Stanford University; California; USA (1971).

[11] George, A.; "Nested Dissection of a Regular Finite element Mesh"; SIAM J. Numer. Anal.; Vol. 10; 345-363 (1973).

[12] George, A. and Liu, J.W.H.; "An Automatic Nested Dissection Algorithm for Irregular Finite Element Problems"; SIAM J. Numer. Anal.; Vol. 15; 1053-1069 (1978).

[13] George, A. and Liu, J.W.H.; "An Implementation of a Pseudoperipheral Node Finder"; ACM Trans. on Math. Software; Vol. 5; 284-295 (1979).

[14] George, A.; "Direct Methods for the Solution of Large Sparse Systems of Linear Systems"; SIAM News; Part I (June 1980) and Part II (July 1980).

[15] George, A. and Liu, J.W.H.; Computer Solution of Large Sparse Positive Definite Systems; Prentice-Hall Inc.; Englewood Cliffs; New Jersey; USA (1981).

[16] Gibbs, N.E., Poole, W.G. and Stockmeyer, P.K.; "An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix"; SIAM J. Numer. Anal.; Vol. 13; 235-251 (1976).

[17] Gibbs, N.E.; "A Hybrid Profile Reduction Algorithm [FI]"; Algorithm 509; ACM trans. on Math. Software; Vol. 2; 378-387 (1976).

[18] Irons, B.M.; "A Frontal Solution Program for Finite Element Analysis"; Inter. J. for Numer. Meth. in Engg; Vol. 2; 5-32 (1970).

[19] King, I.P.; "An Automatic Reordering Scheme for Simultaneous Equations Derived from Network Systems"; Inter. J. for Numer. Meth. in Engg; Vol. 2; 523-533 (1970).

[20] Liu, J.W.H.; "On Reducing the Profile of Sparse Symmetric Matrices"; Ph.D. Thesis; Dept. of Computer Science; University of Waterloo; Ontario; Canada (1976).

[21] Liu, J.W.H. and Sherman, A.H.; "Comparative Analysis of the Cuthill-McKee and the Reverse Cuthill-McKee Ordering Algorithms for Sparse Matrices"; SIAM J. Numer. Anal.; Vol. 13; 198-213 (1976).

[22] MagNet II User's Manual; Computational Analysis and Design Laboratory; Dept. of Elect. Engg; McGill University; Montreal; Canada (1980).

[23] Mishra, M.L.; "A Preconditioned Conjugate Gradient Frontal Solver"; Masters Thesis; Dept. Electrical Engg; McGill University; Montreal; Canada (1981).

[24] Movie.Byu User's Manual; Brigham Young University; USA (1980).

[25] Razzaque, A.; "Automatic Reduction of Frontwidth for Finite Element Analysis"; Inter. J. for Numer. Meth. in Engg; Vol. 15; 1315-1324 (1980).

[26] Silvester, P.P.; "Construction of triangular Finite Element Universal Matrices"; Inter. J. for Numer. Meth. in Engg; Vol. 12; 237-244 (1978).

[27] Silvester, P.P.; "A Practical Manteuffel Spectral Shift Algorithm"; Private Communication (1979).

Appendix A

Basic Matrix Notations and Definitions

Let A be an N by N symmetric positive definite finite element matrix. For the i-th row of A, let

$$f_i(A) = \min \{j : a_{ij} \neq 0\} \text{ and} \qquad (A.1)$$

$$b_i(A) = i - f_i(A). \qquad (A.2)$$

The quantity $f_i(A)$ is simply the column subscript of the leftmost nonzero entry in the i-th row, while $b_i(A)$ is the bandwidth of the i-th row. The bandwidth of A may then be defined as

$$B(A) = \max \{b_i(A) : 1 \leq i \leq N\}. \qquad (A.3)$$

The region (locations) of A within the matrix bandwidth is called the band of A, or

$$\text{Band}(A) = \{\{i,j\} : 0 < i-j \leq B(A)\}. \qquad (A.4)$$

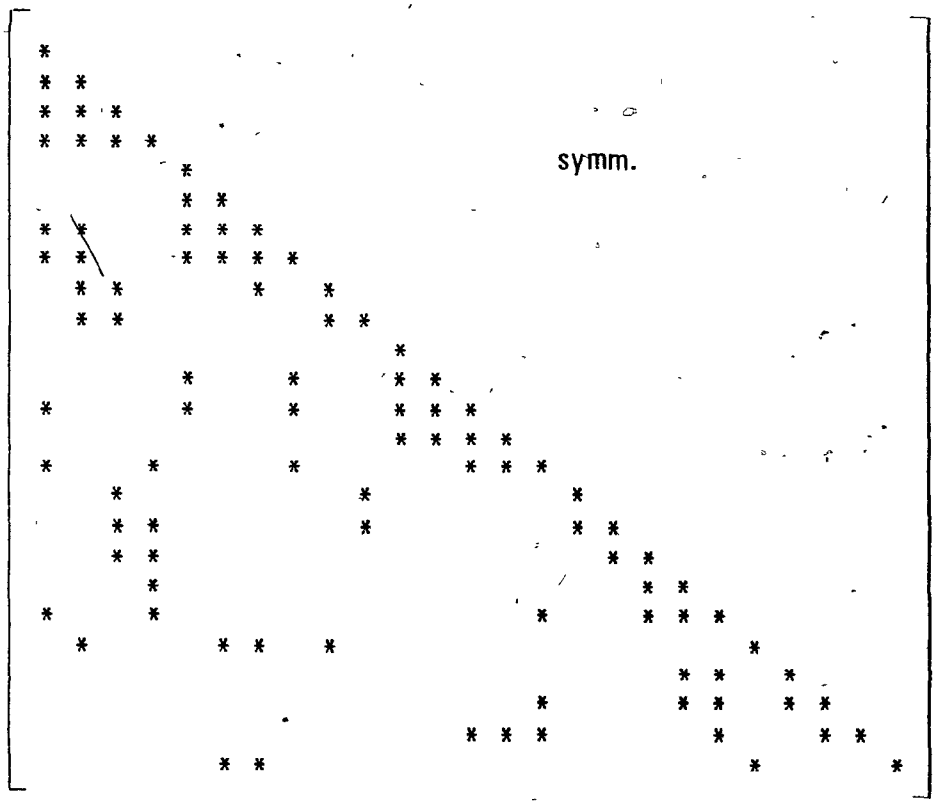
The envelope of the matrix A is the region from the leftmost nonzero f in every row to the main diagonal:

$$\text{Env}(A) = \{\{i,j\} : f_i(A) \leq j < i\}. \qquad (A.5)$$

The envelope size or profile of the matrix A is

$$\text{Profile}(A) = \sum_{i=1}^{N} b_i(A). \qquad (A.6)$$

The matrix of the finite element mesh in figure 2.1 (shown in figure A.1) has a bandwidth of 19, and a profile of 207.



i	f_i	b_i	w_i
1	1	0	8
2	1	1	10
3	1	2	12
4	1	3	12
5	5	0	14
6	5	1	14
7	1	6	13
8	1	7	12
9	2	7	11
10	2	8	10
11	11	0	11
12	5	7	10
13	1	12	10
14	11	3	9
15	1	14	9
16	3	13	8
17	3	14	7
18	3	15	6
19	4	15	6
20	1	19	5
21	2	19	4
22	19	3	3
23	15	8	2
24	13	11	1
25	6	19	0

Fig A.1 The matrix of the finite element mesh in figure 2.1. The nonzero entries are denoted by asterisks.

The frontwidth is an important notion for symmetric matrices. By definition, the i-th frontwidth of the matrix A, denoted by $w_i(A)$, is the number of active rows at the i-th step of the Gaussian elimination. That is, the number of rows in the envelope of A which intersect column i. More formally,

$$w_i(A) = |\{k : k>i \text{ and } a_{K\ell} \neq 0 \text{ for some } \ell \leq i\}|. \quad (A.7)$$

The wavefront or frontwidth of the matrix A is the quantity

$$W(A) = \max \{w_i(A) : 1 \leq i \leq N\}. \quad (A.8)$$

The root-mean-squre (rms) wavefront [8] is

$$W_{rms}(A) = \text{sqrt}\{(\sum_{i=1}^{N} w_i^2(A))/N\}. \quad (A.9)$$

The matrix in figure A.1 has a frontwidth of 14, and a rms wavefront of 9.1673. The profile of the matrix may also be defined as

$$\text{Profile}(A) = \sum_{i=1}^{N} w_i(A). \quad (A.10)$$

In general, the i-th frontwidth of the matrix is different from that of the frontal procedure defined in chapter 2, although both give the number of active matrix rows at the i-th step of elimination. This can be seen by comparing the fourth column of the table in figure A.1 to the last column in the demonstration in chapter 2. The reason is that, in the first case, the Gaussian elimination is applied to the fully assembled matrix (node elimination);

whereas, in the frontal procedure, is applied to partially assembled submatrices in the finite element matrix (element elimination).

An important phenomenon in matrix computation is that of fill-in. It is well known that when the Gaussian elimination is applied to A, the triangular factors L and U may have nonzeros in locations which are zero in A. It is then natural to define the fill of A to be

$$\text{Fill}(A) = \{\{i,j\} : a_{ij} = 0, (L+U)_{ij} \neq 0\}, \qquad \text{(A. 11)}$$

which are the matrix locations where fill-in may occur. This fill-in is confined to the envelope of matrix A [15].