A DIRECT ELEMENT RESEQUENCING PROCEDURE

J. E. Akin
University of Tennessee
Brunel University

R. E. Fulford
University of Tennessee

SUMMARY

Element by element frontal solution algorithms are utilized in many of the existing finite element codes. The overall computational efficiency of this type of procedure is directly related to the element data input sequence. Thus, it is important to have a pre-processor which will resequence these data so as to reduce the element wavefronts to be encountered in the solution algorithm. This paper reports on a direct element resequencing algorithm for reducing the element wavefronts. It also generates computational byproducts that can be utilized in the pre-front calculations and in various post-processors. Sample problems are presented and compared with other algorithms.

INTRODUCTION

Frontal solution procedures for finite element codes were presented independently at about the same time by Hellen [1], Irons [2], and Melosh [3]. These codes utilize an element by element assembly and factorization of the system equations. This procedure has been illustrated for simple models by Hellen [1], and Irons [4]. When utilizing this elimination process one is concerned with the maximum number of active columns or the maximum front associated with any element in the system. This quantity depends solely on the input order of the element incidences cards. By way of comparison, if one were using a frontal solution of the completely assembled system equations, one would be concerned with a wavefront defined by the nodal numbering system. This study describes an algorithm for resequencing the element order so as to reduce the element wavefronts.

THE ELEMENT RESEQUENCING STRATEGIES

Several bandwidth resequencing routines were published before the frontal solution methods became popular. Thus, in investigating frontal reduction methods one should also consider algorithms that were originally written for reducing the bandwidth of a system of equations. Many of these routines are

effective in reducing both the nodal and element wavefronts. The Cuthill-McKee [5] algorithm has been shown to be effective in reducing both the system front and bandwidth [6]. The most efficient nodal resequencing algorithms written specifically for the system frontal method are probably those developed by Levy [7] and King [8]. The Cuthill-McKee algorithm is probably the most commonly used method for reducing system fronts.

The above references are concerned with frontal solutions that are dependent on the nodal numbering system. The present study is directed toward reducing wavefronts encountered in the element by element reduction procedure. Define the front associated with a particular element to be equal to the front of the previous element minus those degrees of freedom that made their last appearance in the previous element plus those degrees of freedom which make their first appearance in the particular element under consideration. This is a quantity that needs to be reduced to save storage and increase the computational efficiency of the equation solving algorithm. The value of the maximum element front is dependent on the input order of the element incidences list. The optimum input sequence is the one that results in the smallest front. Instead it is necessary to utilize a resequencing algorithm to obtain a reduced front in hopes that the reduced value is near the optimum value.

Akin and Pardue [9] have presented two procedures for reducing wavefronts by element resequencing strategies. Both of their procedures were based on generalizations of the Cuthill-McKee method. The disadvantages of these methods are that they require relatively large amounts of storage and initial calculations. The second method has an additional disadvantage in that it does not consider the number of nodes per element (element nodal degree) in its tie breaking options. The present paper introduces a new direct element resequencing strategy that has several advantages over the above methods. First, it requires a much smaller number of initial calculations and storage locations. In addition it has five levels of automatic tie breaking strategies in the resequencing algorithm.

Most resequencing programs are based on the concepts of the level and degree. The present study generalizes these concepts and utilizes the following definitions: Nodes (or elements) adjacent to a given node (or element) are said to be at the same level. The degree of a node (or element) is the number of nodes or elements to which it is connected. The term current degree will denote the degree based on the current number of unresequenced neighbors. For example, if a node has eight element neighbors, three of which have been renumbered, then its element degree is eight and its current element degree would be five.

Consider an effective front defined as, for element i,

$$W_i = W_{i-1} + F_i - L_{i-1}, \quad i = 1, \ldots, NE$$

where W_i is the front width, F_i the number of degrees of freedom first appearing, L_i the number of degrees of freedom making last appearance, NE the number of elements and where $W_0 \equiv L_0 \equiv 0$. As a check on this calculation we know $W_{NE+1} \equiv 0$. The present strategy is based on the concept of minimum element

front growth. That is, select new element i such that the quantity $(F_i - L_{i-1})$ is minimum. Given a starting element as new element one the elements neighboring this element (i.e. at the same 'level') are numbered according to this strategy. Then all elements at the level of new element two are numbered. The numbering continues in this fashion, level by level, until all elements have been numbered. If a starting element is not given, the program could select the one with the smallest or largest F_i.

In such a procedure one often encounters a number of ties of candidates for the next few element numbers. The present code first selects the element (or elements) with the minimum number of new nodes (i.e. the minimum F_i). If this results in a tie then the element with the maximum number of existing nodes is selected (i.e. the maximum L_i). A second tie would be broken by choosing the element with the smallest number of un-numbered element neighbors. If a tie still exists, the element with the largest number of active nodes is selected. Finally, should a tie still exist, the last element in the list is utilized. Clearly, the rank of the tie breaking order could be changed.

EXAMPLE

Clearly one problem is to define F_i and L_i. This is accomplished by defining two scratch arrays, say LFIRST and NOADJL, equal in length to the number of nodes. When LFIRST(J) $\neq$ 0, it equals the new element number in which node J became active. It is initially zero. From this definition one notes that for new element i the value of F_i is equal to the number of nodes on that element for which LFIRST = 0. Once F_i is established the above group of nodes have their zero value of LFIRST changed to i, the new element number. Array NOADJL represents the current number of un-numbered elements adjacent to each node. Initially it equals the total number of elements adjacent to each point. Clearly when NOADJL(K) = 1 then node K is making its last appearance. The value of L_i for element i equals the number of nodes on the element for which NOADJL = 1. Once an element is renumbered the current value of NOADJL for each of its nodes is reduced by one. To illustrate the concepts consider the three-element model shown in Figure 1. The original element wavefronts are 3,4 and 3. The histories of arrays NOADJL and LFIRST are given Table 1. These are established in the following manner:

1. Set new $W_0 = L_0 = 0$, initialize NOADJL, zero LFIRST. Select the new first element L1. Say L1 = 2.

2. The nodes of L1 are 2,3 and 5. We observe:

NODE	ARRAY		COMMENT
2,	LFIRST = 0 ,	new node ;	set LFIRST = 1
	NOADJL = 3 ,	node remains ;	set NOADJL = 2
3,	LFIRST = 0 ,	new node ;	set LFIRST = 1
	NOADJL = 1 ,	node leaves ;	set NOADJL = 0
5,	LFIRST = 0 ,	new node ;	set LFIRST = 1
	NOADJL = 2 ,	node remains ;	set NOADJL = 1

Summary: There are three new nodes, the element front is 3, and one node is leaving, i.e.:

$$F_1 = 3,\ L_1 = 1\ ,\quad W_1 = W_0 + F_1 - L_0 = 0 + 3 - 0 = 3\ .$$

3. The element neighbors of L1 = 2 are elements 1 and 3. Select L2 from that list.

A. Consider candidate element 1: Its nodes are 1, 2, 6.

NODE		COMMENT
1,	LFIRST = 0 ,	new node
	NOADJL = 1 ,	node leaves
2,	LFIRST = 1 ,	active since loop 1
	NOADJL = 2 ,	node remains
6,	LFIRST = 0 ,	new node
	NOADJL = 2 ,	node remains

Summary: 2 new nodes, 1 node leaving, element active since loop 1, 1 old node (3 - 2 = 1)

B. Consider candidate element 3: Its nodes are 2, 5, 6.

NODE	COMMENT
2,	active since loop 1, remains
5,	active since loop 1, leaves
6,	new node, remains

Summary: 1 new node, 1 node leaving, element active since loop 1, 2 old nodes (3 - 1 = 2)

Select L2 = 3. Then $F_2 = 1$, $L_2 = 1$, $W_2 = W_1 + F_2 - L_1 = 3 + 1 - 1 = 3$

For nodes 2, 5, 6, set NOADJL = NOADJL - 1, and if LFIRST = 0, set LFIRST = 2.

4. Are there any un-numbered elements adjacent to L1? Yes, old element 1. Consider element 1: Its nodes are 1, 2, 6.

NODE	COMMENT
1,	is new and leaves
2,	is active since loop 1 and leaves
6,	is active since loop 2 and leaves

Summary: 1 new node, 3 nodes leaving, element active since loop 1, 2 old nodes

Set L3 = 1, $F_3 = 1$, $L_3 = 3$, $W_3 = W_2 + F_3 - L_2 = 3 + 1 - 1 = 3$.

For nodes 1, 2, 5, set LFIRST = 3 if LFIRST = 0, and set NOADJL = NOADJL - 1.

5. No elements remain.

Check calculations: $W_4 = W_3 + F_4 - L_3 = 3 + 0 - 3 = 0$, check!

New element wavefronts are 3, 3, and 3. The new data are shown in Figure 2.

APPLICATIONS

Cuthill [2] has applied various resequencing algorithms to the labelled tree structure shown in Fig. 3. The present algorithm was also applied to this structure and the results are compared with those of Cuthill in Table 2.

The second example test was a simple structure considered by Akhras and Dhatt [10]. It consists of three concentric circles divided, from the center, into eight equal angular segments. Thus it contains eight triangular and sixteen quadrilateral elements. These quadratic elements and their original order are shown in [10]. The third example was a quarter symmetry mesh of a rectangle with a center circular hole. The original element data were generated in a random order so as to cause a large initial wavefront. The fourth problem was a half symmetry shell model. It involved a cylinder with hemispherical caps supported horizontally on two vertical plate saddles. The fifth and sixth problems involved complicated three-dimensional surfaces with branches.

The wavefront reduction data for these problems are given in Table 3. These results show the algorithm to be efficient in reducing the maximum element wavefront. It requires significantly less storage than the algorithm used by Akin and Pardue [9]. Other applications to practical engineering problems have shown reductions of at least thirty percent.

Table 4 shows some relative computation costs. The first item is a measure of the cost of building the neighbors lists that the resequencing subroutine uses as a data base. These calculations are clearly the most expensive. They can be done in various ways and it appears that more efficient procedures can be developed. Much of these data could be utilized in the pre-front stage of the solution algorithm. The second item shows the actual resequencing costs. These are quite small and indicate that one should try several different starting elements since most of the cost goes into the first choice.

Complete program listings and instructions are included in Reference [10].

REFERENCES

1. Hellen, T. K.: A Front Solution for Finite Element Techniques, Central Electricity Generating Board, R & D Dept. RD/B/N 1459, 1969.

2. Irons, B. M.: A Frontal Solution Program for Finite Element Analysis. Intern. J. Num. Meth. Engr. 2, 5-32, 1970.

3. Melosh, R. J. and Bamford, R. M.: Efficient Solution of Load Deflection Equations. J. Structural Div. ASCE, 95, ST4, 661-676, 1969.

4. Irons, B. M. and Kan, K. Y.: Equations Solving Algorithms for the Finite Element Mehtod, Numerical and Computer Methods in Structural Mechanics, S. J. Fenves, et al. (eds.). Academic Press, 497-512, 1973.

5. Cuthill, E. and McKee, J.: Reducing the Bandwidth of Sparse Symmetric Matrices. Proc. ACM Nat. Conf., 157-172, 1969.

6. Cuthill, E.: Several Strategies for Reducing the Bandwidth of Matrices, Sparse Matrices and Their Applications, D. J. Rose and R. D. Willoughby (eds.). Plenum Publishing Co., New York, 157-166, 1972.

7. Levy, R.: Resequencing of the Structural Stiffness Matrix to Improve Computational Efficiency. J. P. L. Quarterly Tech. Review, 1, 2, 61-70, 1971.

8. King, I. P.: An Automatic Reordering Scheme for Simultaneous Equations Derived from Network Analysis. Intern. J. Num. Meth. Engr. 2, 523-533, 1970.

9. Akin, J. E. and Pardue, R. M.: Element Resequencing Algorithm for Frontal Solutions, The Mathematics of Finite Elements and Applications, vol. 2, J. R. Whiteman (ed.). Academic Press, London, 1976.

10. Akhras, G. and Dhatt, G.: An Automatic Node Relabelling Scheme for Minimizing a Matrix Bandwidth. Intern. J. Num. Meth. Engr. 10, 787-797, 1976.

11. Fulford, R. E.: A Wavefront and Bandwidth Reduction Algorithm. M. S. Thesis Dept. Eng. Sci. & Mech., Univ. of Tenn., March, 1977.

Table 1

Logic Arrays for Example

LOOP / J	NOADJL				LFIRST			
	0	1	2	3	0	1	2	3
1	1	1	1	0	0	0	0	3
2	3	2	1	0	0	1	1	1
3	1	0	0	0	0	1	1	1
4	0	0	0	0	0	0	0	3
5	2	1	0	0	0	1	1	1
6	2	2	1	0	0	0	2	2

Table 2

Results for Labelled Tree

Algorithm	Wavefront	Profile
Original	7	107
Cuthill-McKee	9	101
Reverse CMK	4	54
King	5	59
Reverse K	3	38
Levy	2	37
Present	3	42

Table 3

Element Wavefront Reduction Achieved
by the Present Algorithm

Example Number	Number of Nodes	Number of Elements	Types*	Wavefront		
				Original**	Resequenced	Reduction,%
1	24	23	L-2	7	4	43
2	73	24	Q-8,T-6	23	22	4
3	272	121	T-6	123	34	80
4	630	639	Q-8,T-3	224	35	84
5	623	760	Q-4,L-2	230	82	65
6	253	281	Q-4,T-3	52	36	31

* L = Line Element, T = Triangle, Q = Quadrilateral

** Assuming one degree of freedom per node

Table 4

Algorithm Steps as Percent of Total Run Time

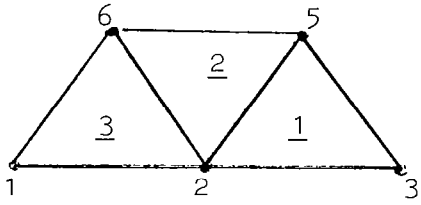
Example	Total CPU Time *	Generate Neighbors List	Resequence Elements **
2	4.35	8.3%	10 %
3	17.47	67.5%	12.5%
4	42.70	80 %	7.5%
5	38.54	75 %	7 %
6	17.55	52 %	15 %

* Seconds on IBM 360/65

** From four different starting elements

Element (L)	Incidences (NODES)	Adjoining Elements (LADJL)
1	1 2 6	2 3
2	2 3 5	1 3
3	2 5 6	1 2

Figure 1. Iron's Element Front Example



Element Number			
old	2	3	1
new	1	2	3

Figure 2. New Element Model

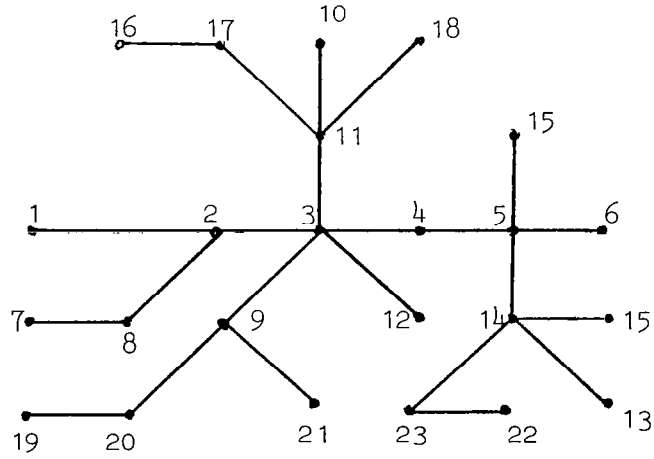


Figure 3. A Labelled Tree [2]