INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 28, 2555–2568 (1989)

THE USE OF PROFILE REDUCTION ALGORITHMS WITH A FRONTAL CODE

I. S. DUFF, J. K. REID AND J. A. SCOTT

Computer Science and Systems Division, Harwell Laboratory, Oxon OX11 0RA, U.K.

SUMMARY

We study profile reduction algorithms when used to order the elements for the frontal solution of a system of linear equations with a symmetric sparsity pattern. We consider two distinct procedures for producing an efficient element ordering; one based on assembling the pattern of the finite-element matrix, reordering the variables and using the new variable order to resequence the elements, and the other based on generating adjacency lists for the elements themselves and reordering the elements directly.

We compare the results of using several variants of these algorithms in conjunction with the Harwell frontal code, MA32, on the CRAY-2 for a range of practical problems. We find that, given suitable enhancements, both approaches are practical and neither is consistently superior to the other.

1. INTRODUCTION

We consider the solution of sparse linear systems of equations

$$\mathbf{Ax}=\mathbf{b} \tag{1}$$

where the $n\times n$ matrix $\mathbf{A}$ is a sum of elemental matrices

$$\mathbf{A}=\sum_{l=1}^{m}\mathbf{A}^{[l]} \tag{2}$$

and the right-hand side vector $\mathbf{b}$ is of the form

$$\mathbf{b}=\sum_{l=1}^{m}\mathbf{b}^{[l]} \tag{3}$$

In equation (2) each matrix $\mathbf{A}^{[l]}$ has entries only in the principal submatrix corresponding to the variables in element l and represents contributions from this element. This principal submatrix is assumed to be dense (any zeros are stored explicitly). The matrix $\mathbf{A}$ may be unsymmetric but the form (2) implies that it has a symmetric sparsity pattern. A frontal method of solution is employed.[4,9,10]

In a finite-element context, the efficiency of a frontal scheme, in terms of both storage and computation time, is dependent upon the ordering of the elements. This is because in the frontal method the system matrix $\mathbf{A}=(a_{ij})$ is never assembled explicitly. The formation of the sum (2) is called the assembly and involves the elementary operation

$$a_{ij}=a_{ij}+a_{ij}^{[l]} \tag{4}$$

Element l contributes to the (i,j)th entry in the matrix $\mathbf{A}$ if and only if variables i and j belong to it.

Received *20 October 1988*
Revised 14 March 1989

An entry a_{ij} is said to be fully summed when all the contributions of the form (4) have been summed. From an inspection of the basic Gaussian elimination operation

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{ik}^{(k)} a_{kk}^{(k)^{-1}} a_{kj}^{(k)} \tag{5}$$

it is clear that the assembly and elimination processes may be interleaved, with each variable being eliminated as soon as its row and column are fully summed, that is after its last occurrence in a matrix $\mathbf{A}^{[l]}$. If this is done, the elimination operations are confined to the submatrix of rows and columns corresponding to variables that have not yet been eliminated but occur in at least one of the elements that have been assembled. This allows all intermediate working to be performed in a full matrix, termed the frontal matrix, whose size increases when a variable appears for the first time and decreases whenever a variable is eliminated. Thus the question under consideration is: for a frontal scheme, what is a good order in which to input the elements?

We now introduce some nomenclature and notation. With reference to equation (1), column j is said to be *active* at stage i if $j \geqslant i$ and there is a non-zero entry in column j with a row index, k, such that $k \leqslant i$. Letting f_i denote the number of columns that are active at stage i, the *maximum wavefront* of $\mathbf{A}$ is given by

$$F = \max_{1 \leqslant i \leqslant n} \{f_i\} \tag{6}$$

The *root-mean-squared* (r.m.s.) *wavefront* is defined to be

$$\tilde{F} = \left(\frac{1}{n}\sum_{i=1}^{n} f_i^2\right)^{1/2} \tag{7}$$

The *profile* of the matrix $\mathbf{A}$ is the total number of coefficients in the lower triangle when any zero ahead of the first entry in its row is excluded. That is,

$$P = \sum_{i=1}^{n} \max_{a_{ij} \neq 0} \{i + 1 - j\} \tag{8}$$

Note that since it is assumed that $\mathbf{A}$ has a symmetric pattern of non-zeros, it follows that

$$P = \sum_{i=1}^{n} f_i \tag{9}$$

The average number of arithmetic operations in a single elimination step in a frontal algorithm is proportional to the mean-squared wavefront and the maximum amount of storage required for the frontal matrix during the Gaussian elimination is dependent upon the maximum wavefront. Moreover, the total storage required and the amount of work involved in the back substitution stage depend on the profile of the matrix. Thus the elements should be numbered in such a way as to reduce F, $\tilde{F}$ and P.

For finite-element problems involving only a small number of elements, it is generally not difficult to order the elements efficiently. Similarly for problems on simple finite-element meshes it is relatively straightforward to order the elements. With more complex grids, this is frequently much harder. Moreover, when finite-element meshes are generated using some automatic procedure, the resulting element numbering may lead to unnecessarily large F, $\tilde{F}$ and P. In such cases, an algorithm to reorder the elements is invaluable. The need for an efficient element ordering is particularly important in non-linear computations where it is necessary to solve a sequence of systems of finite-element equations with the same structure a large number of times.

Various algorithms for automatic ordering of finite elements have been proposed. These include the methods described by Akin and Pardue,[1] Bykat,[2] Razzaque,[12] Pina,[11] Sloan and Randolph,[14] Fenves and Law[6] and Sloan.[13] The different algorithms described in the literature may be broadly divided into two classes. Firstly, there are the methods which relabel the variables and then use the new variable numbers to resequence the elements; the new variable numbers are subsequently discarded. This class of methods will be referred to as indirect element reordering. The other class of methods, which appears to be less popular, reorders the elements directly. In this report we employ the recent algorithm of Sloan[13] to reorder the elements indirectly, and we apply a similar procedure to resequence the elements directly.

In Section 2 we introduce some basic concepts from graph theory then briefly discuss the Harwell profile reduction code MC40, which implements the algorithm due to Sloan.[13] Indirect element-reordering algorithms are discussed in Section 3, and in Section 4 we consider reordering the elements directly. The practical finite-element problems which have been used to test the efficiency of the reordering algorithms and their use in conjunction with the Harwell frontal solver MA32 are described in Section 5, and the numerical results obtained using the CRAY-2 are summarized in Section 6. Finally, in Section 7 concluding remarks and comments are made.

2. THE CODE MC40 FOR PROFILE REDUCTION

The code MC40 is a profile and wavefront reduction code for matrices with a symmetric sparsity pattern. The code was developed by Sloan,[13] who gives full details of the algorithm. We have made some minor modifications to Sloan's algorithm.

Ordering schemes for sparse matrices are related to the labelling of an undirected graph. To describe the code MC40 it is therefore useful to recall some basic definitions from elementary graph theory.

2.1. *Graphs, matrices and finite-element meshes*

An *undirected graph* G is defined to be a pair (V, E), where V is a finite set of *nodes* (or *vertices*), and E is a finite set of *edges* defined as unordered pairs of distinct nodes. A *labelling* (or *ordering*) of a graph $G=(V, E)$ with n nodes is a bijection of $\{1, 2, \ldots, n\}$ onto V. The integer i $(1 \leqslant i \leqslant n)$ assigned to a node in V by a labelling is called the *label* (or *number*) of that node. Two nodes i and j in G are said to be *adjacent* if $(i, j) \in E$. The *degree* of a node $i \in G$ is defined to be the number of nodes in G which are adjacent to i, and the *adjacency list* for i is the list of these adjacent nodes. A *path of length* k in G is an ordered set of distinct nodes $(i_0, i_1, \ldots, i_k)$ where $(i_{j-1}, i_j) \in E$ for $1 \leqslant j \leqslant k$. Two nodes are *connected* if there is a path joining them. A graph G is *connected* if each pair of distinct nodes is connected. Otherwise, G is disconnected and consists of two or more *components*.

The *distance* between nodes i and j in a connected graph G (or in a component of a disconnected graph) is denoted by $d(i, j)$, and is defined to be the number of edges on the shortest path connecting them. The *diameter* $D(G)$ of G is the maximum distance between any pair of nodes. That is,

$$D(G) = \max \{d(i, j) \mid i, j \in V\}$$

Nodes at opposite ends of a diameter of G are known as *peripheral* nodes. A *pseudo-diameter* $\delta(G)$ is defined by any pair of nodes i and j in G for which $d(i, j)$ is close to $D(G)$. A pseudo-diameter may be slightly less than, or equal to, the true diameter and is found by some heuristic algorithm. Nodes defining a pseudo-diameter are termed *pseudo-peripheral* nodes.

A *level structure rooted at a node* r is defined as the partitioning of V into levels $l_1(r)$, $l_2(r), \ldots, l_h(r)$ such that

(i) $l_1(r) = \{r\}$ and

(ii) for $i > 1$, $l_i(r)$ is the set of all nodes that are adjacent to nodes in $l_{i-1}(r)$ but are not in $l_1(r)$, $l_2(r), \ldots, l_{i-1}(r)$.

The level structure rooted at node r may be expressed as the set $L(r) = \{l_1(r), l_2(r), \ldots, l_h(r)\}$, where h is the total number of levels and is termed the *depth*. The *width of level i* is $|l_i(r)|$ (the number of nodes on level i) and the *width* of the level structure is given as

$$w = \max_{1 \leqslant i \leqslant h} \{|l_i(r)|\}$$

We now establish the relationship between graphs and matrices. Let $\mathbf{A} = (a_{ij})$ be an $n \times n$ matrix with a symmetric pattern of entries. Corresponding to $\mathbf{A}$ is a labelled graph G with n nodes and an edge between nodes i and j if and only if $a_{ij} \neq 0$, $i \neq j$. A symmetric permutation of $\mathbf{A}$ leaves its graph unchanged except for the labelling of its nodes.

A *finite-element mesh* is a collection of finite elements in which elements are joined at their common boundaries and vertices. Finite-element nodes may lie at vertices, along the sides, on the faces or within the element itself. Associated with each finite-element node is a set of variables corresponding to the freedoms at that node. The finite-element mesh can be transformed directly into the graph of the assembled finite-element matrix. We will call this the variable connectivity graph to distinguish it from the supervariable connectivity and element connectivity graphs introduced below. The nodes of the variable connectivity graph are the variables defined on the finite-element mesh, and the edges are constructed by making the variables of each element pairwise adjacent.

A *supervariable* is a collection of one or more variables, such that each variable belongs to the same set of finite elements. For example, in the problem involving four 8-node/8-freedom elements shown in Figure 1, variables 1, 2, 6 belong only to element 1 and form a supervariable, and variables 3 and 7 belong to elements 1 and 2 and form another supervariable. Note that, if a node has more than one freedom, they will all belong to the same supervariable. The finite-element mesh can be transformed into a supervariable connectivity graph, whose nodes are the supervariables and whose edges are formed by making the supervariables of each finite element pairwise adjacent. Provided the numbers of the variables in the supervariables are recorded, the supervariable connectivity graph actually provides a compact representation of the variable connectivity graph.

The connectivity of the finite elements may also be represented as a graph; this graph will be termed the element connectivity graph. The nodes in the element connectivity graph correspond to the finite elements in the mesh and the edges describe the interconnection of the finite elements.

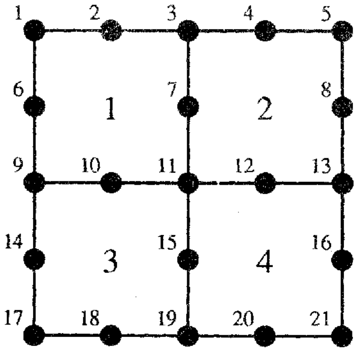

Figure 1. A 4-element problem

2.2. The profile reduction code MC40

The code MC40 may be applied to the variable connectivity graph, the supervariable connectivity graph or the element connectivity graph associated with a finite-element mesh. It will suffice to describe the code for a graph $G=(V, E)$ with N nodes. The graph is stored using an integer array pair (IRN, IP), where the one-dimensional array IRN holds in sequence the adjacency lists for each of the nodes $i \in V$, and IP is an index array of length $N+1$, containing pointers to the beginning of each adjacency list in IRN, with IP $(N+1)$ pointing to one greater than the location in IRN of the last entry in the adjacency list for node N. The fact that G is stored using the array pair (IRN, IP) implies a particular labelling of the associated graph, and this will be referred to as the original labelling.

The reordering comprises two distinct steps:

(i) selection of a pair of pseudo-peripheral nodes and
(ii) node relabelling.

In the first step, for each component of the graph G, a pair of pseudo-peripheral nodes is located. The level structures associated with these nodes are generally deep and narrow, and can be used to find good starting points for profile and wavefront reduction algorithms. The procedure that we use to locate pseudo-peripheral nodes is a modification of that described by Gibbs, Poole and Stockmeyer[8] and George and Liu.[7] To locate a pair of pseudo-peripheral nodes a starting node $s \in G$ of minimum degree is chosen, and the level structure $L(s)$ is generated. The Gibbs, Poole and Stockmeyer algorithm then generates the level structures rooted at each of the nodes in the last level set $l_h(s)$, selected in order of increasing degree. If, for some $r \in l_h(s)$, the depth of $L(r)$ exceeds that of $L(s)$, r replaces s as the starting node, and the procedure is repeated. If no such node r is found, and e is the node in $l_h(s)$ whose associated level structure has the smallest width, the nodes s and e are chosen as pseudo-peripheral nodes. George and Liu[7] modify this algorithm to include a 'short circuiting' strategy by which wide level structures are rejected as soon as they are detected; the same strategy is incorporated within the code MC40. To reduce the number of nodes in the last level set $l_h(s)$ for which it is necessary to generate the rooted level structures, George and Liu[7] and Sloan[13] adopt 'shrinking' strategies. George and Liu choose only one node in $l_h(s)$, and discuss several possible choices for this node. Sloan reports that, for his test problems, using only one node from the last level set could lead to pseudo-peripheral nodes defining a pseudo-diameter which he did not consider to be sufficiently close to the diameter $D(G)$. Sloan chose instead to shrink $l_h(s)$ by taking the first $(m+2)/2$ entries in the sorted list $l_h(s)$ (sorted in ascending sequence of degree), where m is the width of level h. We have found that this may still result in the level structures for a substantial number of nodes having to be constructed. We tried avoiding any node in $l_h(s)$ with a neighbour that had been tested, but found that this was less economical than Sloan's strategy. Instead, we have chosen to construct a list comprising those nodes in $l_h(s)$ with distinct degrees, with ties broken arbitrarily. It was found that, for some problems, this new strategy led to a slight increase in the computed reduced profiles and wavefronts compared with those obtained using Sloan's strategy, but for other problems the effect was a small reduction. For all the problems considered, the use of the new shrinking strategy led to significant reductions in the CPU time required to run MC40 and was therefore adopted in MC40.

In the second step of the algorithm, the nodes in each component are renumbered to obtain a profile which is smaller than that given by the original labelling of the graph. The pseudo-peripheral nodes s and e found in the first step serve as starting and end nodes for the relabelling within their component. The level structure rooted at the end node $L(e)$ is generated and the distance $d(e, i)$ of each node i from the end node is computed. The starting node s is relabelled as node one and a list of nodes that are eligible to receive the next label is formed. At each stage in the relabelling process the list of eligible nodes comprises those nodes which are either adjacent to a node which has been relabelled or are adjacent to a node which is itself adjacent to a relabelled node. The next node to be given a new number is the node among all eligible nodes with the highest priority, where the priority P_i is defined to be

$$P_i = -W_1 * \mathrm{cdeg}(i) + W_2 * d(e, i) \tag{10}$$

for $i \in V(G)$. Here W_1 and W_2 are positive integer weights and the *current degree* cdeg(i) is the potential growth in front size (the number of nodes adjacent to i, not including any node that has been relabelled or is itself adjacent to a relabelled node). The priorities of the eligible nodes are updated at each step. Following Sloan[13] the default values for the weights in MC40 are $W_1 = 2$ and $W_2 = 1$, so that both the front growth and the global structure of the graph are taken into account. Once all the nodes have been assigned new labels, the code checks that the corresponding profile is less than the initial profile. If this is not the case, the original labelling is retained.

3. INDIRECT ELEMENT REORDERING

In the first of the indirect element-reordering algorithms which we consider, the profile reduction code MC40 is applied to the variable connectivity graph. The new ordering of the variable connectivity graph given by MC40 may be used to obtain an efficient frontal solution by relabelling the elements so that the variable-by-variable elimination order is (approximately) conserved. It was observed in the paper by Akin and Pardue[1] that this may be achieved by ordering the elements in ascending sequence of their lowest numbered variable. This procedure is also advocated by Razzaque,[12] Sloan and Randolph[14] and Sloan.[13]

The need to obtain and work with the variable connectivity graph can make this indirect element-reordering algorithm expensive to use. In problems where the finite-element nodes have more than one freedom and in some problems involving high-order elements, the computational cost of resequencing the elements may be considerably reduced if a supervariable connectivity graph is used. If the code MC40 is applied directly to the supervariable connectivity graph then no account is taken of the number of variables in each supervariable. For finite-element grids comprising different element types with multiple freedoms associated with the finite-element nodes, the number of variables in each supervariable may vary significantly. To allow for this variation we need to redefine the degree of a node in a supervariable connectivity graph. In a supervariable connectivity graph, each node is a supervariable and for node i we define the degree to be the total number of variables in the supervariables adjacent to i.

There is a close relationship between the pseudo-diameters of the variable and supervariable connectivity graphs. Suppose that we are given a level structure $\tilde{L}(R)$ of supervariables rooted at supervariable R and that we construct the level structure $L(r)$ of variables rooted at a variable r belonging to R. The variables belonging to the supervariables in level $\tilde{l}_i(R)$ will also lie in level $l_i(r)$ $(i = 2, 3, \ldots)$. In fact, the only difference between the level sets will be that the variables of R apart from r will be in level set 2. It follows that the diameters of the two graphs are identical. Unfortunately, the details of the code do not mean that the same pseudo-diameter is necessarily chosen, but the lengths of the pseudo-diameters are likely to be the same. However, we may certainly use the supervariable representation as a mechanism to construct a pseudo-diameter for the variable connectivity graph.

Similarly, there is a close relationship between the orderings on the variable and supervariable connectivity graphs. Apart from one detail, we could use the supervariable representation to construct the ordering for the variables. The detail is that, when a variable is chosen, all the other variables of its supervariable have current degree zero and are likely to be chosen next, but they need not be. Of course, in the supervariable representation, all the variables in a supervariable are chosen together. Modifying the variable ordering to give top priority to variables with current degree zero is anyway very sensible, so we feel justified in regarding the whole of the supervariable ordering scheme as an efficient means of implementing variable ordering. Several details of the coding mean that we rarely get identical results, however.

Once the supervariable version of MC40 has been employed to order the supervariables, the finite elements are ordered in ascending sequence of their lowest numbered supervariable. For many problems the number of nodes and edges in the supervariable connectivity graph is considerably less than the corresponding number in the variable connectivity graph and, since the time taken to run MC40 is dependent upon the number of nodes and edges in the graph, using the supervariable connectivity graph instead of the variable connectivity graph can result in significant savings in terms of both execution time and storage. This is illustrated in Section 6.

4. DIRECT ELEMENT REORDERING

An alternative approach to renumbering the variables (or supervariables) and using the new labelling of the variable (or supervariable) connectivity graph to reorder the elements is to reorder the elements directly. There are several possible approaches to this problem. Direct element reordering is discussed briefly by Akin and Pardue.[1] The algorithms described by Bykat,[2] Pina[11] and Fenves and Law[6] are direct element-reordering schemes. In particular, the procedure proposed by Bykat is a modification of the Cuthill–McKee algorithm (Cuthill and McKee[3]) for reducing the bandwidth of a sparse matrix with a symmetric sparsity pattern, applied to the element connectivity graph instead of the variable connectivity graph. Bykat generates an element connectivity graph by defining two elements to be adjacent to one another whenever they share a common edge and describes his algorithm in detail for planar triangular elements.

Fenves and Law[6] generalize the definition of adjacent elements used by Bykat to problems in n dimensions, $n=1, 2, 3$. Fenves and Law define two elements in n dimensions to be adjacent whenever they possess a common boundary of $(n-1)$ dimensions. Thus in three dimensions two volumetric elements are adjacent if they share a common two-dimensional boundary face; in two dimensions planar elements are interconnected by one-dimensional boundary lines; and one-dimensional finite elements are adjacent if they have a common finite-element node. Fenves and Law employ this definition of adjacent elements to generate an element connectivity graph, and reorder the elements directly by applying the standard Cuthill–McKee algorithm.

It was noted by Fenves and Law that the adjacency of elements cannot always be completely represented by the above definition of adjacent elements, since n-dimensional elements are not necessarily connected through $(n-1)$-dimensional boundaries. In addition, adjacent finite elements do not necessarily have the same geometric dimensions. In such examples, the element connectivity graph may become disconnected, and each component must be numbered independently. This contributes to the difficulties associated with attempting to implement this algorithm.

We have adopted a simpler procedure for reordering the elements directly than that suggested by Bykat or by Fenves and Law: we define two elements to be adjacent to each other whenever they have one or more variables in common. Using this definition, it is not difficult to generate the associated element connectivity graph; the user does not need to provide information on the different types of elements in the grid other than a list of the variables associated with each finite element (which is exactly what the Harwell frontal solver MA32 requires). Once the element connectivity graph has been generated, the code MC40 may be employed to reorder the elements directly.

The definition of element adjacency used by Fenves and Law[6] was considered for the test problems with finite-element meshes in which all the elements were of the same geometric dimension. The computed maximum and r.m.s. wavefronts using this approach were marginally smaller for some problems compared with those obtained using our simple definition of element adjacency, but for other problems the reverse was true. For problems involving different types of elements, using the Fenves and Law definition is complicated since it is necessary to distinguish between elements of different dimensions and for such problems we employed only our simple definition of element adjacency.

The main disadvantage of reordering the finite-element mesh directly using the element connectivity graph in which two elements are adjacent whenever they share a common variable is that the number of variables in each element is not taken into consideration. It is anticipated that the algorithm will perform at its best when the number of variables per element is relatively constant. To allow for finite-element meshes comprising finite elements with different numbers of freedoms we have considered modifying the code MC40 when it is applied to the element connectivity graph. In the second stage of MC40, we define the priority of node (element) i to be

$$P_i = -W_1 * \mathrm{cadj}(i) + W_2 * d(e, i) + W_3 * N_i \tag{11}$$

where cadj(i) is the number of elements adjacent to i, not including those which have been relabelled or are themselves adjacent to a relabelled element, and N_i is the number of variables in element i. The values assigned to the weights determine the importance of each of these criteria. It was found that a suitable choice for the weights is $W_1 = 12$, $W_2 = 6$ and $W_3 = 1$, which essentially means that W_3 affects only the breaking of ties in Sloan's priority function. For each of our test problems the r.m.s. wavefront obtained using the third weight W_3 was less than or equal to that found using the unmodified MC40 code ($W_3 = 0$). Results using this 'weighted' direct element-reordering algorithm, which will be referred to as the direct element-reordering algorithm I, are included in Section 6.

The direct element-reordering algorithm I aims to reduce the number of elements which are active during the frontal method of solution, where an element is active if it has been assembled but not all its variables have been eliminated. Reducing the number of elements that are active will indirectly reduce the number of active variables. In an attempt to reduce both the number of active elements and the number of active variables, we have considered a slightly different direct element-reordering algorithm that uses the element connectivity graph together with the lists of supervariables associated with each element. The priority of element i is now defined to be

$$P_i = -W_1 * \mathrm{ngain}(i) + W_2 * d(e, i) - W_3 * \mathrm{nadj}(i) \tag{12}$$

where ngain(i) is the number of variables element i will introduce into the front less the number that can then be eliminated, and nadj(i) is the number of elements adjacent to element i which have not yet been relabelled. It was observed that the choice $W_1 = 12$, $W_2 = 6$ and $W_3 = 1$ gave good element orderings and these values were chosen as the default values for the weights. If assembling element i leads to the elimination of a variable v, then ngain(i) = cdeg(v), where cdeg(v) is defined as in equation (10). In this case, the priority function (12) is therefore Sloan's function with a third weight to resolve ties. If every element leads to an elimination, we have another implementation of the Sloan variable ordering (with another tie-breaking strategy). In general, however, this will not be the case and the algorithm is therefore different but closely related. This reordering algorithm will be referred to as the direct element-reordering algorithm II.

5. THE TEST PROBLEMS

Ten problems are used to compare the element-reordering algorithms outlined in Sections 3 and 4. The problems all arise from practical applications. A brief description of each problem is given in Table I. Problems 1–4 and 8–10 were taken from the Harwell–Boeing sparse matrix collection.[5] In each example, we were given a list of the unknowns for each element in the finite-element mesh. These lists do not include the constrained variables which lie on boundaries with Dirichlet boundary conditions. Using these lists the element orderings produced by the reordering algorithms will differ slightly from those which would be obtained if complete lists of the variables associated with each element in the finite-element mesh were available. The number of supervariables in each problem was not given but was obtained from the given data when the supervariable connectivity graph was generated.

Table I. The test problems (CEGB = Central Electricity Generating Board; TPD = Theoretical Physics Division, Harwell; LSMC = Lockheed Space and Missiles Company)

Problem	Origin	Description	Number of variables	Number of super-variables	Number of elements	Elements
1	CEGB	3D model of a turbine blade	2694	316	108	20-node/60-freedom bricks
2	CEGB	Framework problem from structural engineering	3222	537	791	2-node/12-freedom bars
3	CEGB	3D model of a cylinder with a flange	2859	483	128	20-node/60-freedom bricks
4	CEGB	2D cross-section of a reactor core	2996	1418	551	8-node/16-freedom quadrilaterals
5	TPD	2D groundwater flow problem	2254	2123	571	9-node/9-freedom quadrilaterals
6	TPD	2D groundwater flow problem	2656	2467	620	3-node/3-freedom bars
					1	6-node/6-freedom triangles
					79	9-node/9-freedom quadrilaterals
7	TPD	3D groundwater flow problem	19197	16426	1848	27-node/27-freedom bricks
					770	18-node/18-freedom tetrahedrals
8	LSMC	2D model of a component used in ocean-mining	691	176	72	2-node/6-freedom bars
					158	2-node/12-freedom bars
					94	3-node/18-freedom triangles
9	LSMC	2D model of part of a vehicle	3416	702	72	2-node/6-freedom bars
					10	2-node/12-freedom bars
					48	3-node/18-freedom triangles
					556	4-node/24-freedom quadri-laterals
10	LSMC	Framework model of a launch umbilical tower	2208	368	944	2-node/12-freedom bars

6. NUMERICAL RESULTS

The results obtained by applying the different element-reordering algorithms to the test problems outlined in Section 5 are presented in this section. Tables II and III illustrate the maximum and r.m.s. wavefronts, respectively, using the Sloan indirect reordering algorithm applied to the variable connectivity graph and to the supervariable connectivity graph (with the appropriate modification to MC40), and using the direct element-reordering algorithms I and II. For comparison the maximum and r.m.s. wavefronts for the user-supplied element ordering are included. Note that, if the maximum wavefront is F, the maximum storage required for the frontal matrix is F^2 in the unsymmetric case and $F(F+1)/2$ in the symmetric case.

The reductions which are achieved in the maximum and r.m.s. wavefronts using the reordering algorithms are obviously dependent upon the original user-supplied element order. It is clear that,

Table II. Maximum wavefronts for the different reordering algorithms

	Algorithm				
		Indirect reordering		Direct reordering	
Problem	User-supplied ordering	Variables	Super-variables	Algorithm I	Algorithm II
1	120	120	120	120	117
2	354	78	114	186	78
3	348	285	291	309	291
4	152	124	92	112	130
5	202	50	50	59	59
6	177	25	23	29	25
7	995	1136	1070	890	661
8	316	99	123	159	114
9	834	266	181	278	217
10	1266	60	60	114	72

Table III. Root-mean-squared wavefronts for the different reordering algorithms

	Algorithm				
		Indirect reordering		Direct reordering	
Problem	User-supplied ordering	Variables	Super-variables	Algorithm I	Algorithm II
1	91·2	89·7	91·2	92·6	89·6
2	245·9	60·0	73·4	126·7	60·3
3	216·2	196·8	190·7	187·7	190·5
4	108·3	77·6	61·5	76·7	84·5
5	137·0	35·0	35·8	36·3	36·6
6	121·0	21·0	20·9	21·0	20·9
7	725·2	814·3	824·1	637·4	520·4
8	151·5	66·8	77·6	100·9	72·1
9	583·0	138·1	118·0	153·4	134·0
10	748·0	45·8	45·8	86·7	50·2

for some of the test problems, in particular problems 1 and 3, the user was able to provide a reasonable element ordering but for most of the problems this was not the case, and the reordering algorithms were able to provide significant improvements in the maximum and r.m.s. wavefronts. For problem 7, the indirect reordering algorithms produced a worse element ordering than that supplied by the user. It should be noted that the reordering algorithms do not guarantee a reduction in the maximum and r.m.s. wavefronts in the frontal solver.

The CPU times required by the element-reordering algorithms for each of the test problems are shown in Table IV. These statistics are for the CRAY-2. For each of the algorithms, the CPU times are given for generating the appropriate connectivity graph and running the profile reduction code MC40. In addition, the total CPU time required to reorder the elements (including the time taken to generate the connectivity graph and run MC40), the CPU time taken by the Harwell frontal solver MA32 using the new element order, and the total CPU time to reorder the elements and run the frontal solver MA32 once are given. For comparison, the CPU times for MA32 using the user-supplied element orderings are included. As can be seen from Table IV, the most expensive procedure considered for reordering the elements is, as predicted, the Sloan indirect reordering algorithm using the variable connectivity graph. Considerable savings are, however, achieved in the computation times for the indirect reordering algorithm using supervariables in place of variables.

7. CONCLUSIONS

In this report we have illustrated the use of two distinct classes of methods for resequencing finite elements prior to running the Harwell frontal solver MA32. The first class of methods comprises indirect element-reordering schemes based on Sloan's reordering algorithm. The second class of methods applies a modification of Sloan's algorithm to the element connectivity graph and directly resequences the elements. The indirect element-reordering algorithms aim to reduce directly the number of columns which are active during any stage of the frontal method by reordering the elements in such a way as to mimic the desired order of elimination of the variables. In contrast, by running Sloan's profile reduction code on the element connectivity graph, the direct reordering algorithm I aims to reduce the number of elements which are active during the frontal method of solution thereby indirectly reducing the number of active variables. The direct element-reordering algorithm II attempts to directly reduce both the number of active elements and the number of active variables, without first resequencing the variables.

As expected, there is little to choose between the qualities of the orderings produced by the variable and supervariable connectivity graphs (Tables II and III), while the time for supervariable reordering algorithm is consistently less (Table IV). Of these two, we therefore prefer the supervariable reordering algorithm.

For problems involving solid finite elements, the maximum and r.m.s. wavefronts achieved by the direct element reordering algorithm II were generally, but not consistently, smaller than those obtained by the direct algorithm I. However, algorithm I performed poorly on framework problems (problems 2 and 10). Therefore, although it is faster to resequence the elements using the somewhat simpler algorithm I in place of II, we conclude that the direct element-reordering algorithm II should be employed in preference to I.

For most of the problems considered, the maximum and r.m.s. wavefronts achieved by the indirect reordering algorithm using the supervariables and the direct element-reordering algorithm II were similar and, for a given test problem, it was not possible to predict which of the two methods would yield the smallest values. For the test problems with fewer supervariables than finite elements, the indirect method was the faster method, while for problems in which the number

Table IV. CPU times (seconds) on the CRAY-2 for reordering the elements and running the Harwell frontal solver MA32

Problem	Algorithm	CPU time (seconds)				
		Generate connectivity graph	MC40	Reorder elements	MA32	Total
1	User-supplied element order	—	—	—	0·96	0·96
	Indirect reordering using variables	0·82	3·00	3·83	0·93	4·76
	Indirect reordering using supervariables	0·04	0·05	0·10	0·96	1·06
	Direct reordering algorithm I	0·03	0·01	0·04	1·00	1·04
	Direct reordering algorithm II	0·03	0·02	0·05	0·93	0·98
2	User-supplied element order	—	—	—	2·50	2·50
	Indirect reordering using variables	0·28	1·28	1·59	0·71	2·30
	Indirect reordering using supervariables	0·06	0·04	0·12	0·76	0·90
	Direct reordering algorithm I	0·04	0·10	0·14	1·16	1·30
	Direct reordering algorithm II	0·04	0·10	0·15	0·71	0·86
3	User-supplied element order	—	—	—	1·80	1·80
	Indirect reordering using variables	0·98	4·13	5·13	1·78	6·91
	Indirect reordering using supervariables	0·07	0·14	0·22	1·73	1·95
	Direct reordering algorithm I	0·03	0·02	0·05	1·70	1·75
	Direct reordering algorithm II	0·03	0·04	0·07	1·73	1·80
4	User-supplied element order	—	—	—	0·96	0·96
	Indirect reordering using variables	0·27	1·58	1·87	0·79	2·66
	Indirect reordering using supervariables	0·11	0·32	0·44	0·64	1·08
	Direct reordering algorithm I	0·04	0·09	0·13	0·78	0·91
	Direct reordering algorithm II	0·04	0·13	0·18	0·80	0·98
5	User-supplied element order	—	—	—	1·11	1·11
	Indirect reordering using variables	0·37	0·65	1·18	0·64	1·82
	Indirect reordering using supervariables	0·11	0·53	0·66	0·64	1·30
	Direct reordering algorithm I	0·03	0·07	0·10	0·67	0·77
	Direct reordering algorithm II	0·03	0·11	0·14	0·67	0·81

6	User-supplied element order	—	—	—	1·73	1·73
	Indirect reordering using variables	0·46	0·60	1·25	0·77	2·02
	Indirect reordering using supervariables	0·13	0·51	0·66	0·77	1·43
	Direct reordering algorithm I	0·03	0·08	0·15	0·76	0·91
	Direct reordering algorithm II	0·03	0·12	0·16	0·76	0·92
7	User-supplied element order	—	—	—	90·0	90·0
	Indirect reordering using variables	5·06	58·7	62·8	115·0	179·8
	Indirect reordering using supervariables	2·82	40·2	45·1	112·0	152·1
	Direct reordering algorithm I	0·41	1·75	2·55	48·7	51·3
	Direct reordering algorithm II	0·41	3·01	3·51	44·1	47·6
8	User-supplied element order	—	—	—	0·35	0·35
	Indirect reordering using variables	0·13	0·32	0·46	0·19	0·66
	Indirect reordering using supervariables	0·03	0·02	0·06	0·24	0·30
	Direct reordering algorithm I	0·02	0·08	0·10	0·19	0·29
	Direct reordering algorithm II	0·02	0·11	0·13	0·22	0·35
9	User-supplied element order	—	—	—	10·45	10·45
	Indirect reordering using variables	0·64	2·63	3·31	1·41	4·72
	Indirect reordering using supervariables	0·09	0·10	0·21	1·14	1·35
	Direct reordering algorithm I	0·06	0·12	0·18	2·10	2·29
	Direct reordering algorithm II	0·06	0·13	0·19	1·45	1·64
10	User-supplied element order	—	—	—	11·15	11·15
	Indirect reordering using variables	0·33	0·90	1·27	0·57	1·84
	Indirect reordering using supervariables	0·07	0·03	0·12	0·53	0·65
	Direct reordering algorithm I	0·05	0·15	0·20	0·73	0·93
	Direct reordering algorithm II	0·05	0·17	0·23	0·59	0·82

of supervariables exceeded the number of elements the direct method proved the faster method. We conclude from our empirical evidence that the user may wish to reorder the finite elements using either the indirect algorithm applied to the supervariables or the direct algorithm II. A code which implements both the indirect algorithm applied to the supervariables and the direct algorithm II is to be included in the Harwell Subroutine Library as routine MC43.

ACKNOWLEDGEMENTS

The authors would like to thank S. W. Sloan for helpful discussions. The authors are also grateful to C. P. Jackson and T. E. Preece for providing the data for the test problems 5, 6 and 7.

REFERENCES

1. J. E. Akin and R. M. Pardue, 'Element resequencing for frontal solutions', in J. R. Whiteman (ed.), *Mathematics of Finite Elements and Applications*, Academic Press, New York, 1975.
2. A. Bykat, 'A note on an element ordering scheme', *Int. j. numer. methods eng.*, **11**, 194–198 (1977).
3. E. Cuthill and J. McKee, 'Reducing the bandwidth of sparse symmetric matrices', *Proc. 24th National Conference of the Association for Computing Machinery*, Brandon Press, New Jersey, 1969, pp. 157–172.
4. I. S. Duff, 'Enhancements to the MA32 package for solving sparse unsymmetric matrices', *Harwell Report AERE R11009*, HMSO, London, 1983.
5. I. S. Duff, R. G. Grimes and J. G. Lewis, 'Sparse matrix test problems', *ACM Trans. Math. Softw.*, **15**, 1–14 (1989).
6. S. J. Fenves and K. H. Law, 'A two-step approach to finite element ordering', *Int. j. numer. methods eng.*, **19**, 891–911 (1983).
7. A. George and W. H. Liu, 'An implementation of a pseudoperipheral node finder', *ACM Trans. Math. Softw.*, **5**, 284–295 (1979).
8. N. E. Gibbs, W. G. Poole, Jr. and P. K. Stockmeyer, 'An algorithm for reducing the bandwidth and profile of a sparse matrix', *SIAM J. Numer. Anal.*, **13**, 236–250 (1976).
9. P. Hood, 'Frontal solution program for unsymmetric matrices', *Int. j. numer. methods eng.*, **10**, 379–400 (1976).
10. B. M. Irons, 'A frontal solution program for finite-element analysis', *Int. j. numer. methods eng.*, **2**, 5–32 (1970).
11. H. L. Pina, 'An algorithm for frontwidth reduction', *Int. j. numer. methods eng.*, **17**, 1539–1546 (1981).
12. A. Razzaque, 'Automatic reduction of frontwidth for finite element analysis', *Int. j. numer. methods eng.*, **15**, 1315–1324 (1980).
13. S. W. Sloan, 'An algorithm for profile and wavefront reduction of sparse matrices', *Int. j. numer. methods eng.*, **23**, 239–251 (1986).
14. S. W. Sloan and M. F. Randolph, 'Automatic element reordering for finite-element analysis with frontal schemes', *Int. j. numer. methods. eng.*, **19**, 1153–1181 (1983).