RAL-TR-98-031

On ordering elements for a frontal solver.*

by

Jennifer A. Scott

Abstract

The efficiency of the frontal method for the solution of finite-element problems depends on the order in which the elements are assembled. We look at using variants of Sloan's algorithm to reorder the elements. Both direct and indirect reordering algorithms are considered and are used in combination with spectral orderings. Numerical experiments are performed on a range of practical problems and, on the basis of the results, we propose a hybrid element resequencing algorithm for use with a frontal algorithm.

Keywords: ordering finite-elements, frontal method, Sloan algorithm, spectral method.

AMS(MSC 1991) subject classifications: 65F05, 65F50.

CR classification system: G.1.3.

---

* Current reports available by anonymous ftp from `matisa.cc.rl.ac.uk` in the directory `pub/reports`. This report is in file sRAL98031.ps.gz.

Department for Computation and Information,
Atlas Centre, Rutherford Appleton Laboratory,
Oxon OX11 0QX, England.

March 31, 1998.

Contents

1 Introduction

In this report, we are interested in the efficient use of the frontal method to solve large sparse systems of linear equations

$$\mathbf{AX} = \mathbf{B}, \tag{1.1}$$

where the $n \times n$ matrix $\mathbf{A}$ is a sum of $nelt$ finite-element matrices

$$\mathbf{A} = \sum_{l=1}^{nelt} \mathbf{A}^{(l)}, \tag{1.2}$$

and the $n \times nrhs$ matrix $\mathbf{B}$ of right-hand sides is of the form

$$\mathbf{B} = \sum_{l=1}^{nelt} \mathbf{B}^{(l)}. \tag{1.3}$$

Each matrix $\mathbf{A}^{(l)}$ has nonzeros only in a few rows and columns and corresponds to the matrix from element l. The frontal method is a variant of Gaussian elimination, the main feature of the method being that the contributions $\mathbf{A}^{(l)}$ from the finite-elements are assembled one at a time and the construction of the assembled coefficient matrix $\mathbf{A}$ is avoided by interleaving assembly and elimination operations. An assembly operation is of the form

$$a_{ij} \Leftarrow a_{ij} + a_{ij}^{(l)}, \tag{1.4}$$

where $a_{ij}^{(l)}$ is the (i,j)th nonzero entry of the element matrix $\mathbf{A}^{(l)}$. A variable is *fully summed* if it is involved in no further sums of the form (1.4) and is *partially summed* if it has appeared in at least one of the elements assembled so far but is not yet fully summed. The Gaussian elimination operation

$$a_{ij} \Leftarrow a_{ij} - a_{il}[a_{ll}]^{-1}a_{lj} \tag{1.5}$$

may be performed once all the terms in the triple product in (1.5) are fully summed.

Since variables can only be eliminated after they are fully summed, the assembly order will determine, to a large extent, the order in which the variables are eliminated. At any stage during the assembly and elimination processes, the fully and partially summed variables are held in an in-core *frontal matrix*. Dense linear algebra operations are performed on the frontal matrix. For efficiency, in terms of both storage and arithmetic operations, the elements must be assembled in an order that keeps the size of the frontal matrix, known as the *wavefront*, as small as possible. Of interest is

- the maximum wavefront, since this affects the in-core storage needed,
- the sum of the wavefronts, known as the *profile*, since this determines the total storage needed for the matrix factors, and

- the root-mean-square wavefront, since the work performed when eliminating a variable is proportional to the square of the current wavefront.

In the past, a number of algorithms for automatically ordering finite elements have been proposed (for example, Akin and Pardue, 1975, Bykat, 1977, Razzaque, 1980, Pina, 1981, Sloan and Randolph, 1983, Fenves and Law, 1983, Sloan, 1986, Duff, Reid and Scott, 1989, Kaveh, 1991, and Paulino, Menezes, Gattass and Mukherjee, 1994). Duff et al. divide these algorithms into direct and indirect element ordering algorithms. Direct algorithms order the elements directly while indirect algorithms use a two-step approach in which the variables are first relabelled and used to resequence the elements; the new variable indices are subsequently discarded. Duff et al. report that both approaches can be used effectively and neither has been found to be consistently superior to the other.

The Harwell Subroutine Library code `MC43` (Duff et al., 1989) implements both a direct and an indirect ordering algorithm, based on the profile reduction algorithm of Sloan (1986). Motivated by the findings of Kumfert and Pothen (1997), we recently looked at a number of ways of improving the performance and efficiency of Sloan's algorithm (Reid and Scott, 1998). These included implementing the priority queue as a binary heap and using a hybrid algorithm that combines a spectral ordering (see, for example, Barnard, Pothen and Simon, 1995) with the Sloan algorithm. This work led to improved codes for profile reduction, `MC60` and a driver `MC61`, being included in the Harwell Subroutine Library (1995) and prompted us to look at revising `MC43` in a similar way. The new element ordering code is called `MC63`.

The outline of this report is as follows. In Section 2, we briefly review graphs that can be associated with a finite-element mesh. These graphs are fundamental to our reordering algorithms. In Section 3, we look at Sloan's algorithm. In Section 4, we discuss using spectral orderings to resequence elements and introduce a hybrid method that combines using a spectral ordering with Sloan's algorithm. The design of our new code `MC63` is discussed in Section 5. Numerical experiments on a range of practical problems are reported on in Section 6. Results illustrating the use of `MC63` with the Harwell Subroutine Library frontal solver `MA62` are given Section 7, and some concluding comments are made in Section 8. Specification sheets for `MC63` are included in the Appendix.

2 Finite element graphs

The element resequencing algorithms that we use in this report are based on the method of (Sloan, 1986, 1989). The method, which has been widely used during the last decade for profile reduction, exploits the close relationship between a matrix $\mathbf{A} = \{a_{ij}\}$ of order n with a symmetric sparsity pattern and its undirected graph with n nodes. Two nodes i and j are neighbours (or are adjacent) in the graph if and only if a_{ij} is nonzero. 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 with its degrees of freedom can be transformed into the graph of the assembled finite-element matrix and, for convenience, we call this the *variable connectivity* graph. 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.

In many finite-element problems, there are a number of freedoms at each node of the finite-element mesh. Moreover, nodes may belong to the same set of elements. Both features can be exploited through the use of supervariables. A *supervariable* is a collection of one or more variables, such that each variable belongs to the same set of finite elements. 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 list of variables in each supervariable is recorded, the supervariable connectivity graph provides a compact representation of the variable connectivity graph. For problems in which the number of supervariables is substantially less than the number of variables, Sloan's algorithm is much more efficient if supervariables are used. Reid and Scott (1998) report results that illustrate this.

For finite element problems, Sloan's method may also be applied to the element connectivity graph, in which the nodes are the finite elements. There is more than one way in which the element connectivity may be defined. Bykat (1977) generates the 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. This definition was generalized by Fenves and Law (1983) to problems in k dimensions ($k = 1, 2, 3$), by defining two elements in k dimensions to be adjacent whenever they possess a common boundary of $(k-1)$ dimensions. The resulting graph is termed the *dual graph* (see Paulino et al., 1994). The main advantage of the dual graph is its economy in terms of data storage because the number of edges is generally substantially fewer than in the variable or supervariable graphs. A disadvantage is that the adjacency of elements cannot always be completely represented by this definition of adjacent elements, since k-dimensional elements are not necessarily connected through $(k-1)$-dimensional boundaries. In addition, adjacent finite elements do not necessarily have the same dimensionality. In such examples, the dual graph may become disconnected, and each component must be numbered independently. This contributes to the difficulties associated with attempting to implement this algorithm.

A more convenient way of defining element adjacency is to define two elements to be adjacent whenever they have one or more variables in common. The resulting graph is the *element communication* graph and was used by Duff et al. (1989) and Paulino et al. (1994). Throughout the remainder of this report, the element connectivity graph will refer to the element communication graph.

3 Sloan's algorithm

In this section, we give a brief outline of Sloan's algorithm for profile reduction and discuss how the method can be extended for element reordering. Here and elsewhere we assume that the variable connectivity and the element connectivity graphs are connected. If not, it is straightforward to apply the algorithm to each component, and all our software allows for this.

3.1 The basic algorithm

Sloan's algorithm for reordering the nodes of a connected graph has two distinct phases:

1. Selection of a start node and an end node.
2. Node reordering.

In the first phase, the start and end nodes are chosen to be the endpoints of a pseudodiameter. Sloan finds a pseudodiameter using a modified version of the Gibbs, Poole and Stockmeyer (1976) algorithm. In the second phase, the pseudodiameter is used to guide the reordering. One end s of the pseudodiameter is used as the start node and the other e is used as the target end node. Sloan ensures that the position of a node in his ordering is not far from one for which the distance from the target end node is monotonic decreasing. He is able to improve the profile and wavefront by localized reordering. Sloan begins at the start node s and uses the priority function

$$P_i = -W_1 c_i + W_2 d(i,e) \tag{3.1}$$

for node i, where W_1 and W_2 are integer weights, c_i (the *current degree*) is the amount that the wavefront will increase if node i is numbered next, and $d(i,e)$ is the distance to the target end node. At each stage, the next node in the ordering is chosen from a list of eligible nodes to maximize P_i. The list of eligible nodes comprises the neighbours of nodes that have already been numbered and their neighbours. A node has a high priority if it causes either no increase or only a small increase to the current wavefront and is at a large distance from the target end node. Thus, a balance is kept between the aim of keeping the number of nodes in the front small and including nodes that have been left behind (further away from the target end node than other candidates).

Following numerical experimentation, Sloan recommends the pair (2,1) for the weights. However, the results of Kumfert and Pothen (1997) and Reid and Scott (1998) indicate that, for some problems, there are considerable advantages in using other values. In particular, the choice (16,1) can yield much smaller profiles. To allow the user to experiment with different choices of the weights, the new profile reduction code `MC60` of Reid and Scott has weights that are input parameters.

3.2 Sloan's algorithm for indirect element ordering

Sloan's algorithm may be used to reorder elements by applying the method to the variable connectivity graph and then resequencing the elements in ascending order of their earliest variable in the new variable order. In most finite-element problems, the number of supervariables (see Section 2) is significantly less than the number of variables. In such cases, it is more efficient to apply a modified version of the Sloan algorithm to the supervariable connectivity graph. The modifications to the Sloan algorithm take into account the number of variables associated with each supervariable (see Duff et al., 1989).

3.3 Sloan's algorithm for direct element ordering

An alternative approach to element reordering is to apply Sloan's algorithm directly to the element connectivity graph. The main disadvantage of this is that the number of variables in each element is not taken into consideration. To allow for finite-element meshes comprising finite elements with different numbers of freedoms, Duff et al. (1989) looked at modifying the priority function in the second phase of the algorithm. An element is said to be "active" if it has been assembled but has one or more unassembled neighbours. In an attempt to reduce both the number of elements that are active at each stage of the frontal method and the number of partially summed variables, Duff et al. define the priority of element i to be

$$P_i = -W_1 ngain_i + W_2 d(i,e) - W_3 nadj_i. \tag{3.2}$$

Here $ngain_i$ 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 that have not yet been relabelled. The weights used by Duff et al. in the Harwell Subroutine Library code `MC43` are $(10, 5, 1)$. If assembling element i leads to the elimination of a single variable j, then $ngain_i = c_j$, where c_j is defined as in equation (3.1). In this case, the priority function (3.2) is Sloan's function with a third weight to resolve ties. If every element leads to such an elimination, we have another implementation of the Sloan variable ordering algorithm (with a tie-breaking strategy). In general, however, this will not be the case and the algorithm is therefore different but closely related.

4 Spectral and hybrid reordering algorithms

4.1 Spectral reordering

Spectral algorithms have been used in recent years for matrix profile and wavefront reduction. Barnard et al. (1995) describe a spectral algorithm that associates a Laplacian matrix $\mathbf{L}$ with the given matrix $\mathbf{A}$ with a symmetric

sparsity pattern,

$$\mathbf{L} = \{l_{ij}\} = \begin{cases} -1 & \text{if } i \neq j \text{ and } \quad a_{ij} \neq 0 \\ 0 & \text{if } i \neq j \text{ and } \quad a_{ij} = 0 \\ \sum_{i \neq j} |l_{ij}| & \text{if } i = j. \end{cases} \tag{4.1}$$

An eigenvector corresponding to the smallest positive eigenvalue of the Laplacian matrix is termed a *Fiedler vector*. The spectral permutation of the variables is computed by sorting the components of a Fiedler vector into monotonically nonincreasing or nondecreasing order.

For unassembled finite-element problems, the new variable order can be used to obtain an element ordering. We refer to this as the *indirect spectral* element reordering algorithm. Alternatively, Paulino et al. (1994) propose constructing the Laplacian matrix associated with the element connectivity graph and reordering the elements by sorting the components of a Fiedler vector of this Laplacian. The results presented by Paulino et al. suggest that the method can be effective for finite-element problems but comparisons were only reported with the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms (Lewis, 1982). In our numerical experiments (see Section 6), we call this method the *direct spectral* element reordering method.

4.2 The hybrid method

Kumfert and Pothen (1997) observe that spectral orderings do well in a global sense but are often poor locally. They therefore propose using the spectral method to find a global ordering that guides the second phase of Sloan's method. Their results show that this can yield a final ordering with a much smaller profile than using either the spectral method alone or Sloan's method using the Gibbs-Poole-Stockmeyer pseudodiameter. Further experiments by Reid and Scott (1998) support this view, particularly for very large problems. The so-called *hybrid method* uses a priority function in which the distance $d(i, e)$ from the target end node is replaced by p_i, the position of node i in the spectral ordering. Specifically, for a graph with n nodes, Reid and Scott (1998) use the priority function

$$P_i = -W_1 c_i - W_2 (h/n) p_i, \tag{4.2}$$

where h is the number of level-sets in the level set structure rooted at the first node. Reid and Scott recommend computing orderings for the pairs of weights $(1, 2)$ and $(16, 1)$ and choosing the one with the smallest profile. This is the default in the driver `MC61`.

Kumfert and Pothen (1997) and Reid and Scott (1998) use the hybrid method to reorder assembled matrices. In the present study, we are concerned with unassembled matrices. We can extend the hybrid method to this class of problems in one of two ways:

1. In the indirect algorithm, use the hybrid method to order the variables. We will refer to this method as the *indirect hybrid* algorithm. In practice, for efficiency, the spectral variable ordering is mapped to a spectral supervariable ordering and the supervariable connectivity graph is used.

2. Apply the spectral method to the element connectivity graph. In the second phase of Sloan's algorithm, replace (3.2) with a priority function

$$P_i = -W_1 ngain_i + W_2(h/nelt)pelt_i - W_3 nadj_i \tag{4.3}$$

where $nelt$ is the number of elements, h is the number of level-sets in the level set structure rooted at the first element, and $pelt_i$ is the position of element i in the direct element spectral ordering. We will call this the *direct hybrid* algorithm.

We remark that, although in our experiments we only use the spectral orderings in the hybrid algorithms, any input ordering can be used. Our software is written to allow this.

5 Software design

In this section, we discuss the design of our new package, `MC63`, for ordering finite elements. Our new subroutines are named according to the naming convention of the Harwell Subroutine Library (1995).

There are three entries to `MC63`:

- `MC63I` sets default values for the control parameters. It should normally be called once prior to calling `MC63A`. The control parameters include stream numbers for diagnostic printing and parameters that determine whether or not supervariables are to be used and whether the user wishes to supply a global priority function.

- `MC63A` reorders the elements. The user chooses whether a direct or an indirect algorithm is implemented.

- `MC63B` computes, for a given element order, the maximum wavefront, the profile, and the root-mean-square wavefront. An option exists for checking the input data.

Full details of the calling sequence and the argument lists are given in the specification sheets (see Appendix). Note that we work only with the pattern of the matrix. Thus for matrices that are not positive definite, the actual factorization may be more expensive and require more storage than is indicated by `MC63B`. We now look in more detail at the reordering routine `MC63A`.

`MC63A` accepts lists of variables belonging to the elements and, after performing initial checks on the user's data, calls `MC63B` to compute statistics for the natural element order 1, 2,.., $nelt$. At this point, `MC63B` also checks the element variable lists for out-of-range and duplicated indices. If such entries are found they are either removed and the computation continues after issuing a warning message or terminates if this has been requested.

If supervariables are wanted, they are constructed using the Harwell Subroutine Library profile reduction package `MC60`. Otherwise, each variable is treated as a supervariable. The element variable lists are overwritten by

element supervariable lists. A map of variable to supervariable indices allows the user to later restore the element variable lists, if desired.

For each supervariable, the number of elements involving it is counted. Lists of the elements associated with the supervariables are then constructed. If the user has selected a direct element reordering algorithm, the element connectivity graph is constructed from the supervariable lists, otherwise the supervariable connectivity graph is constructed from the element lists.

In the indirect element reordering algorithm, `MC60C` is used to reorder the supervariables. The user may optionally specify a global priority vector whose components p_i are used in the priority function (4.2). Once the supervariables have been reordered, the elements are resequenced in ascending order of their earliest supervariable in the new supervariable order. The new supervariable indices are not preserved.

In the direct element ordering algorithm, the element connectivity graph is relabelled using either the priority function (3.2) or (4.3). If (3.2) is used, the start and target end nodes (s, e) are computed using the modified Gibbs-Poole-Stockmeyer algorithm (MGPS) of Reid and Scott (1998). Again, `MC60` is used for this. To use (4.3), the user must supply an element global priority vector *pelt*.

Having chosen the start and target end elements, the start element is numbered first and a list of elements that are eligible to be numbered next is formed. When selecting the element with highest priority for renumbering next from the list of eligible elements, a simple sequential search is performed while the list is less than a given threshold and a switch to a binary heap search is made once the list exceeds this threshold. As in `MC60`, a threshold of 100 is used. Management of the list of eligible elements is discussed in detail by Reid and Scott (1998).

A final call to `MC63B` (without error checking) computes statistics for the new element order.

We remark that, in `MC63`, equations (3.2) and (4.3) do not define the priority function fully since we give maximum priority to any element that will introduce no new variables into the front.

6 Numerical results

In this section, we first describe the problems that we use for testing the element reordering algorithms discussed in this report and then present numerical results.

6.1 Test problems

Each of the test problems arises from a real engineering or industrial application. A brief description of each problem is given in Table 6.1. All but the last four problems were supplied as unassembled finite-element problems. The first five problems are taken from the Harwell-Boeing Collection (Duff, Grimes and Lewis, 1992). Unfortunately, the number of unassembled element problems included in either the Harwell-Boeing Collection or the Matrix

Identifier	Degrees of freedom	Number of super-variables	Number of elements	Description/discipline
LOCK2232	2208	368	944	Launch umbilical tower
LOCK3491	3416	702	684	Cross-cone vehicle structure
MAN5976	5882	2399	784	Deformation of 3D cylinder
CEGB3306	3222	537	791	2.5D framework problem
CEGB3024	2996	1418	551	2D reactor core section
RAMAGE02	1476	4939	1400	Navier Stokes and continuity equations
AEAC5081	5801	1637	800	Double glazing problem
TRDHEIM	22098	2868	813	Mesh of the Trondheim fjord
TSYL201	20685	2881	960	Part of oil production platform
OPT1	15449	3802	977	Part of oil production platform
CRPLAT2	18010	3004	3152	Corrugated plate field
CHAM	12834	12834	11070	Part of an engine cylinder
TUBU	26573	26573	23446	Engine cylinder model
BCSPWR10	5300	5291	7232	Eastern US power network
BCSSTK15	3948	3948	12992	Model of an off-shore platform
BCSSTK18	11948	11444	21056	Nuclear power station
BCSSTK32	44609	43342	78779	Automobile chassis

Table 6.1: The test problems

Market (http://math.nist.gov/MatrixMarket/) is limited and those that are available all are small by today's standards. We have selected the unassembled problems from the Harwell-Boeing Collection with at least 500 elements and have additionally used some test examples that are not included in the Collection. Problem AEAC5801 came from Andrew Cliffe of AEA Technology and RAMAGE02 from Alison Ramage of the University of Strathclyde. Problems TRDHEIM, CRPLAT2, OPT1, and TSYL201 were supplied by Christian Damhaug of Det Norske Veritas Research, Norway, and CHAM and TUBU were from Ron Fowler of the Rutherford Appleton Laboratory. For CHAM and TUBU, only lists of supervariables belonging to each element were available so for these problems the number of variables is equal to the number of supervariables. To provide further test examples with a large number of elements, we took four problems from the Harwell-Boeing Collection (BCSPWR10, BCSSTK15, BCSSTK18, and BCSSTK32) that are supplied in assembled form and used the Harwell Subroutine Library (HSL) code `MC37` to generate a set of element matrices that, if assembled, would yield the same matrix.

When testing the element ordering algorithms, the elements were input in random order. Our old code `MC43` and the new code `MC63` are written in standard Fortran 77, and all the results presented in this section were obtained using the EPC (Edinburgh Portable Compilers, Ltd) Fortran 90 compiler with optimization -O running on a 143 MHz Sun Ultra 1. In our experiments involving the spectral method, the Fiedler vector was obtained using Chaco

2.0 (Hendrickson and Leland, 1995). We used the SymmLQ/RQI option and the input parameters were chosen to be the same as those used by Kumfert and Pothen (1997). Note that we do not include timings for the hybrid methods because the Chaco package is written in C and the HSL does not currently have a Fortran code for computing the Fiedler vector.

6.2 `MC43` versus `MC63`

In Tables 6.2 and 6.3, we compare the performance of the old code `MC43` with that of the new code `MC63`. Results are given for both the direct and indirect algorithms, using the weights $(10, 5, 1)$ and $(2, 1)$, respectively. In Table 6.2, the maximum and root-mean-square wavefronts are given, and in Table 6.3, timings are presented. As expected, the codes generally yield orderings of

Identifier	`MC43`				`MC63`			
	Direct		Indirect		Direct		Indirect	
LOCK2232	72	48.5	60	45.8	72	48.2	60	45.8
LOCK3491	209	135.5	181	118.1	203	126.5	266	137.5
MAN5976	204	171.1	230	178.1	213	174.7	256	181.3
CEGB3306	78	60.3	114	73.9	78	60.4	78	60.3
CEGB3024	124	76.0	92	61.2	124	76.0	92	63.4
RAMAGE02	1452	1289.3	1502	1333.3	1452	1289.3	1472	1328.5
AEAC5081	149	180.7	190	119.1	156	180.2	175	116.5
TRDHEIM	348	172.4	324	139.1	348	172.4	324	146.3
TSYL201	540	511.2	534	511.2	540	511.2	696	505.3
OPT1	1006	619.9	883	544.1	1006	619.3	804	530.1
CRPLAT2	538	376.3	560	328.0	392	292.8	470	358.9
CHAM	412	333.0	412	313.1	412	331.1	412	332.9
TUBU	638	407.2	863	449.4	630	406.6	848	444.3
BCSPWR10	69	45.9	81	42.3	70	46.2	72	40.9
BCSSTK15	424	282.8	354	234.1	365	260.0	329	227.4
BCSSTK18	541	320.1	506	294.6	562	321.7	500	304.2
BCSSTK32	3280	1910.0	1682	840.7	2628	1655.2	999	572.0

Table 6.2: The maximum and root-mean-square wavefronts found by `MC43` and `MC63`.

comparable quality, although the new code performed significantly better than the old code on the last example. The differences are attributable to the differences in the implementations of the algorithms. For example, the two codes handle supervariables in a slightly different manner. `MC63` takes the numbers of variables in the supervariables into account when calculating the width of a level-set structure but only `MC43` allows for the numbers of variables in the supervariables when calculating the degrees of the supervariables in the list of potential start nodes. The new code also uses a slightly different modification of the Gibbs-Poole-Stockmeyer algorithm when choosing start and end nodes. This is discussed in detail by Reid and Scott (1998).

Identifier	MC43		MC63	
	Direct	Indirect	Direct	Indirect
LOCK2232	0.03	0.03	0.03	0.03
LOCK3491	0.03	0.03	0.03	0.03
MAN5976	0.04	0.11	0.05	0.10
CEGB3306	0.02	0.02	0.02	0.02
CEGB3024	0.02	0.05	0.03	0.06
RAMAGE02	0.27	0.75	0.21	0.59
AEAC5081	0.50	0.60	0.50	0.60
TRDHEIM	0.08	0.13	0.09	0.13
TSYL201	0.10	0.16	0.11	0.16
OPT1	0.13	0.32	0.12	0.32
CRPLAT2	0.18	0.18	0.19	0.18
CHAM	1.75	1.34	0.97	0.79
TUBU	4.41	3.22	2.70	1.62
BCSPWR10	0.18	0.16	0.15	0.13
BCSSTK15	2.79	0.43	1.17	0.33
BCSSTK18	2.62	1.15	1.01	0.51
BCSSTK32	35.65	14.25	10.16	4.88

Table 6.3: A comparison of CPU times for `MC43` and `MC63`. (Sun Ultra).

Reid and Scott (1998) found that which end of the pseudodiameter is chosen as the start node can have an effect on the final ordering. They concluded that there is generally a slight advantage in choosing the pseudoperipheral node that gives the narrowest width as the start node and this choice is used by `MC63`, but not by `MC43`. To try and account for why `MC63` performs particularly well on problem BCSSTK32, we tried not swapping the ends of the pseudodiameter. In this case, the maximum and root-mean-square wavefronts for the direct algorithm increased to 3325 and 1924.9, respectively, and for the indirect algorithm they rose to 1631 and 834.3, respectively, making the results for the old and new codes similar.

Many of our test problems are not large enough to illustrate the time savings that can be achieved by using a binary heap to manage the queue of eligible nodes. Sloan (1986) observed that the binary heap search is the method of choice when the root-mean-square wavefront exceeds several hundred nodes and for smaller problems a simple sequential search is faster. The method we use of commencing with code that performs a simple search, and switches to code that uses a binary heap if the number of eligible nodes exceeds a threshold, ensures `MC63` is as efficient as `MC43` on small problems, but is substantially faster on large problems (see, for example, problems TUBU, BCSSTK18 and BCSSTK32).

6.3 Adjusting the weights

In this section, we consider the effect of adjusting the weights in the priority function. As already mentioned, Duff et al. recommend the weights $(10, 5, 1)$ but Kumfert and Pothen (1997) suggest that, for some problems, other values

give much better results. In our first test, we compare using $W_3 = 1$ in the direct reordering priority functions (3.2) and (4.3) with $W_3 = 0$. The weights (W_1, W_2) are given the values of $(10, 5)$ for the Sloan method and, following Reid and Scott (1998), $(5, 10)$ for the hybrid method. Our findings are presented in Table 6.4. The results suggest that for Sloan there is a slight advantage in using a third weight to resolve ties but, in general, the difference in the root-mean-square wavefront between using $W_3 = 0$ and $W_3 = 1$ is small (less than 2 per cent). For the hybrid method, the results do not support the use of a third weight.

Identifier	Sloan		Hybrid	
	$W_3 = 0$	$W_3 = 1$	$W_3 = 0$	$W_3 = 1$
LOCK2232	51.3	48.2	61.8	61.8
LOCK3491	125.5	126.5	208.0	213.2
MAN5976	179.6	174.7	174.5	174.3
CEGB3306	59.8	60.3	65.3	65.3
CEGB3024	73.5	76.1	49.3	49.7
RAMAGE02	1289.3	1289.3	1321.9	1385.8
AEAC5081	117.4	108.2	108.3	108.2
TRDHEIM	172.4	172.4	148.1	149.6
TSYL201	511.2	511.2	511.5	511.6
OPT1	621.6	619.3	537.5	536.4
CRPLAT2	292.1	292.8	238.0	238.0
CHAM	330.6	331.1	329.6	332.6
TUBU	411.8	406.6	403.6	408.3
BCSPWR10	46.3	46.2	34.3	34.5
BCSSTK15	269.7	260.0	188.8	205.6
BCSSTK18	336.1	321.7	209.5	221.3
BCSSTK32	1493.6	1655.2	756.4	828.5

Table 6.4: Root-mean-square wavefronts with $W_3 = 0, 1$.

We have examined the wavefronts for the direct and indirect ordering algorithms for $(W_1, W_2) = (5w_1, 5w_2)$, with (w_1, w_2) equal to each of the 13 pairs $(1, 64)$, $(1, 32)$, $(1, 16)$, ..., $(1, 1)$, $(2, 1)$, ..., $(64, 1)$ on all the test matrices. Our findings are shown in Table 6.5. In this table we show the percentage increases in the root-mean-square wavefront from the best value when the recommended weights of $(10, 5, 1)$ for Sloan and $(1, 2, 0)$ for the hybrid method are used. We see that, in general, the recommended weights give root-mean-square wavefronts that are within 5 per cent of the minimum value. For each method there are a small number of problems for which weights other than the recommended ones give significant improvements. However, a closer look at the results reveals that the weights that give minimum wavefronts differ with the problem and method. For example, direct Sloan applied to TRDHEIM has the smallest root-mean-square wavefront when $(w_1, w_2) = (1, 2)$ but for problem BCSSTK32, the minimum is achieved with the pair $(1, 16)$. To allow the user to experiment with different weights, in `MC63` the weights are input parameters

Identifier	Sloan		Hybrid	
	Direct	Indirect	Direct	Indirect
LOCK2232	0.0	0.0	0.0	0.0
LOCK3491	0.0	10.3	8.5	0.0
MAN5976	5.7	0.0	2.4	0.3
CEGB3306	0.6	0.3	3.9	3.5
CEGB3024	0.4	0.0	1.3	0.8
RAMAGE02	2.3	0.1	5.0	16.7
AEAC5081	0.4	0.0	0.1	0.1
TRDHEIM	25.3	0.1	2.6	0.0
TSYL201	0.0	0.7	0.0	0.0
OPT1	11.3	1.4	11.1	0.0
CRPLAT2	0.6	9.5	0.2	0.7
CHAM	0.6	0.0	0.4	3.0
TUBU	0.0	0.0	4.6	2.4
BCSPWR10	20.7	15.8	8.5	5.4
BCSSTK15	0.7	0.0	0.2	0.6
BCSSTK18	3.4	25.1	0.0	0.0
BCSSTK32	32.6	0.0	10.7	3.1

Table 6.5: Percentage increases in the root-mean-square wavefront above the minimum value when the recommended weights are used.

under the user's control.

6.4 Sloan versus hybrid

In Table 6.6, we present root-mean-square wavefronts for the different algorithms discussed in this report. The weights recommended in Section 6.3 are used. For purposes of comparison, we include results for the original element order. We note that, in most instances (not including the last four problems that were artificially "disassembled" using `MC37`), this order was thought, by the originator of the problem, to be a "good" element order. In Table 6.6, we highlight in bold the smallest root-mean-square wavefront for each problem and any within 2 per cent of the smallest. We see that, if the elements are not originally well ordered, both the direct and indirect spectral algorithms can substantially reduce the root-mean-square wavefront. However, comparison of columns 3 and 6 and columns 4 and 7 demonstrate that it is worthwhile to use Sloan's method to refine the spectral ordering. By looking also at Table 6.5, we observe that the only problems where the hybrid method gives poorer results than the corresponding spectral method are those for which the recommended weights give a root-mean-square wavefront that is far from the minimum. For these problems, the hybrid method becomes competitive if other weights are used. For example, if we use the weights $(1, 1)$ in the indirect hybrid algorithm in place of the recommended values of $(1, 2)$, the root-mean-square wavefront for RAMAGE02 reduces to 1298.9, and this is smaller than the indirect spectral root-mean-square wavefront.

Identifier	Original	Spectral		Sloan		Hybrid	
	order	Direct	Indirect	Direct	Indirect	Direct	Indirect
LOCK2232	74.8	74.3	65.1	48.2	**45.6**	61.8	48.6
LOCK3491	583.0	227.4	128.0	126.5	135.4	208.0	**103.7**
MAN5976	175.9	182.7	185.3	**174.7**	180.7	**174.5**	**174.4**
CEGB3306	245.9	100.4	95.0	**60.3**	**60.5**	65.3	64.9
CEGB3024	108.3	52.9	53.2	76.1	63.3	**49.7**	**49.4**
RAMAGE02	1492.3	1335.2	1403.1	**1289.3**	1328.4	1321.9	1515.6
AEAC5081	142.2	129.4	**108.2**	**108.2**	116.5	**108.2**	**108.4**
TRDHEIM	181.9	163.3	156.0	172.4	**146.3**	**148.1**	**144.8**
TSYL201	861.6	530.0	513.6	**511.2**	**505.3**	**511.5**	**502.7**
OPT1	2067.7	573.7	604.9	619.3	**530.1**	536.4	557.0
CRPLAT2	1178.1	257.2	251.7	292.8	358.7	**238.0**	**242.0**
CHAM	769.0	346.1	353.6	**331.1**	**332.9**	**329.5**	**338.9**
TUBU	1298.6	470.1	460.9	406.6	447.2	403.6	**393.2**
BCSPWR10	1458.5	44.4	39.7	46.2	40.9	34.3	**32.4**
BCSSTK15	282.2	214.5	190.8	260.0	227.4	188.8	**175.5**
BCSSTK18	547.9	239.5	239.5	321.7	304.3	209.5	**200.6**
BCSSTK32	2472.8	720.1	704.3	1655.2	572.0	756.4	**537.9**

Table 6.6: Root-mean-square wavefronts with different algorithms.

A comparison between the Sloan and hybrid algorithms is less clear-cut. The hybrid method is primarily intended for very large problems and, on the basis of the results we have obtained, the hybrid method generally out-performs Sloan for problems with a large number of elements. We conclude from our empirical evidence that the user may wish to reorder the finite elements using either a direct or an indirect algorithm and may wish to use a hybrid method. `MC63` allows each of these options to be selected.

7 Element orderings and frontal solvers

We have looked at using variants of Sloan's algorithm to reorder finite-elements. Both direct and indirect reordering algorithms have been considered and used in combination with spectral orderings. As discussed in the introduction, the main motivation behind this work was the need for element orderings that are efficient when used with a frontal solver. In this section, we present results of using the `MC63` element orderings with a frontal solver. In the Harwell Subroutine Library we have two frontal codes for real matrices: `MA42` (Duff and Scott, 1996) for general unsymmetric problems and `MA62` (Duff and Scott, 1997) for symmetric positive-definite systems. Both codes are designed for unassembled finite-element matrices, although `MA42` does include an option for entering the assembled matrix row-by-row. The matrix factors may optionally be held in direct access files. For efficiency, Level 3 BLAS are used in the innermost loop of the matrix factorization.

The element ordering schemes we have considered work only with the pattern of the finite element matrices. They are therefore most useful for

Identifier	Factorization time (seconds)		Solve time (seconds)		Number of ops ($*10^7$)		Storage (Kwords)	
	Before	After	Before	After	Before	After	Before	After
LOCK2232	1.1	0.32	0.03	0.02	5.1	0.7	218	120
LOCK3491	4.0	0.99	0.11	0.06	51.3	7.2	1123	448
MAN5976	33.9	33.1	0.85	0.82	482.1	471.2	9133	8982
CEGB3306	0.64	0.45	0.03	0.03	3.1	1.7	232	219
CEGB3024	0.51	0.46	0.04	0.04	2.6	2.4	249	233
RAMAGE02	232.6	175.4	2.2	1.8	3811.9	2852.3	25547	21847
AEAC5081	1.5	1.1	0.09	0.07	12.4	7.5	786	585
TRDHEIM	5.7	4.6	0.34	0.32	51.8	36.6	2872	2389
TSYL201	98.1	38.3	1.5	1.0	1528.4	535.5	17236	10654
OPT1	54.5	31.6	0.9	0.7	857.1	437.4	10031	7596
CRPLAT2	126.5	11.3	1.7	0.49	1918.9	112.6	17985	4583
CHAM	53.9	13.1	0.95	0.46	766.2	151.7	9689	4472
TUBU	121.9	35.6	1.7	1.1	1858.9	440.0	17277	10742

Table 7.1: The results of using `MC63` with the frontal solver `MA62`. (CRAY J932)

positive definite matrices. For more general matrices, the need to preserve numerical stability may lead to the actual factorization being more expensive and requiring more storage. To illustrate the effectiveness of our element ordering algorithms, in this section we present results for `MA62`, using the user-supplied element order and the `MC63` element order. As discussed in Section 5, `MC63` offers both a direct and indirect version of Sloan's algorithm and a direct and indirect hybrid algorithm. We saw in Table 6.6 that the method that gives the smallest root-mean-square wavefront is problem dependent so we have chosen the best ordering for each problem (using the recommended weights). Default values are used for all `MA62` control parameters.

The experimental results quoted in Table 7.1 were obtained on a single processor of a CRAY J932 using 64-bit floating-point arithmetic, and the vendor-supplied BLAS. The codes `MC63` and `MA62` were compiled using the CRAY Fortran compiler f90, with default options. For each problem, values for the entries of the matrix were generated using the HSL pseudo-random number generator `FA01`. The times include the i/o overhead for using direct access files. The storage is the total storage for the matrix factors and includes both real and integer storage. Since, on the CRAY, both integers and reals are stored in 64-bit words, the value is just the sum of the number of real and the number of integer words needed. The number of floating-point operations ("ops") counts all operations (+,-,*,/) equally. The "Solve" times quoted are for a single right-hand side.

The results demonstrate the importance of reordering the elements and illustrate that for problems that are not initially well ordered, substantial savings can be achieved by using `MC63`. Where there is a significant reduction in the root-mean-square wavefront it is reflected in much lower factorization and solve times, operation counts, and factor storage, although we note that the effect of using Level 3 BLAS means that the poorer orderings can have a

higher Megaflop rate so that, for some problems, the ratio of times, before and after ordering, is not as high as the operation count ratio. Furthermore, MA62 is partly able to offset the effect of a poor ordering by exploiting zeros within the frontal matrix (see Duff and Scott, 1997 and Cliffe, Duff and Scott, 1997). For example, we note that, for the problem CEGB3306, the root-mean-square wavefront is reduced by a factor of 4 (see Table 6.6) using MC63 but the saving in factor storage is small. This is because the root-mean-square wavefront assumes that the frontal matrix is dense but MA62 is able to take some advantage of zeros in the front.

8 Conclusions

In this report, we have looked at using variants of Sloan's algorithm to reorder finite-elements for use with a frontal solver. Both direct and indirect versions of the reordering algorithm have been considered and used in combination with spectral orderings. We found that, in general, there is little difference in the quality of the ordering obtained using the direct or the indirect method: for some problems the direct method gives the best reduction in the wavefront but for others the converse is true. Our results suggest that the hybrid method is superior to the spectral method and generally out-performs Sloan for large problems but offers no consistent advantage for smaller problems. Further tests involving problems with a large number of elements are needed before firm conclusions can be drawn.

A disadvantage of the hybrid method is the need to compute a global priority function. The time taken to compute a spectral ordering is significantly more than that needed to compute start and end nodes for the Sloan algorithm (see Kumfert and Pothen, 1997). For this reason, if the tradeoff between the quality of the ordering and the time taken for computing the ordering favours fast reordering algorithms, the Sloan algorithm may be preferred, with the direct method selected if the number of elements is less than the number of supervariables and the indirect method used otherwise. However, in our experiments with the frontal method, the time required to compute an element ordering was small compared with that needed by the matrix factorization and solve steps. For large problems, it may therefore be worthwhile experimenting with the options offered by MC63 before using the frontal solver, particularly if a number of factorizations are to use the same element ordering.

The code MC63 is available and will be included in the next release of the Harwell Subroutine Library. Anyone interested in using the code should contact the author for details of price and conditions of use.

9 Acknowledgements

I would like to thank my colleagues John Reid and Iain Duff at the Rutherford Appleton Laboratory their interest and comments on this report.

References

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, 1975.

S.T. Barnard, A. Pothen, and H. Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numerical Linear Algebra with Applications*, **2**, 317–198, 1995.

A. Bykat. A note on an element ordering scheme. *Inter. Journal on Numerical Methods in Engineering*, **11**, 194–198, 1977.

K.A. Cliffe, I.S. Duff, and J.A. Scott. Performance issues for frontal schemes on a cache-based high performance computer. Technical Report RAL-TR-97-001, Rutherford Appleton Laboratory, 1997. To appear in *Inter. Journal on Numerical Methods in Engineering.*

I.S. Duff and J.A. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Mathematical Software*, **22**(1), 30–45, 1996.

I.S. Duff and J.A. Scott. MA62 – a new frontal code for sparse positive-definite symmetric systems from finite-element applications. Technical Report RAL-TR-97-012, Rutherford Appleton Laboratory, 1997.

I.S. Duff, J.K. Reid, and J.A. Scott. The use of profile reduction algorithms with a frontal code. *Inter. Journal on Numerical Methods in Engineering*, **28**, 2555–2568, 1989.

I. S. Duff, R. G. Grimes, and J. G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (Release I). Technical Report RAL-TR-92-086, Rutherford Appleton Laboratory, 1992.

S.J. Fenves and K.H. Law. A two-step approach to finite element ordering. *Inter. Journal on Numerical Methods in Engineering*, **19**, 891–911, 1983.

N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM J. Numerical Analysis*, **13**, 236–250, 1976.

Harwell Subroutine Library. *A Catalogue of Subroutines (Release 12).* Advanced Computing Department, AEA Technology, Harwell Laboratory, Oxfordshire, England, 1995.

B. Hendrickson and R. Leland. The Chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1995.

A. Kaveh. A connectivity coordinate system for node and element ordering. *Computer Structures*, **41**, 1217–1223, 1991.

G. Kumfert and A. Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT*, **18**, 559–590, 1997.

J.G. Lewis. Implementation of the Gibbs-Poole-Stockmyer and Gibbs-King algorithms. *ACM Trans. Mathematical Software*, **8**, 180–189, 1982.

G.H. Paulino, I.F. Menezes, M. Gattass, and S. Mukherjee. Node and element resequencing using the Laplacian of a finite element graph: Part II – implementation and numerical results. *Inter. Journal on Numerical Methods in Engineering*, **37**, 1531–1555, 1994.

H.L. Pina. An algorithm for frontwidth reduction. *Inter. Journal on Numerical Methods in Engineering*, **17**, 1539–1546, 1981.

A. Razzaque. Automatic reduction of frontwidth for finite element analysis. *Inter. Journal on Numerical Methods in Engineering*, **15**, 1315–1324, 1980.

J.K. Reid and J.A. Scott. Ordering symmetric sparse matrices for small profile and wavefront. Technical Report RAL-TR-98-016, Rutherford Appleton Laboratory, 1998.

S.W. Sloan. An algorithm for profile and wavefront reduction of sparse matrices. *Inter. Journal on Numerical Methods in Engineering*, **23**, 1315–1324, 1986.

S.W. Sloan. A Fortran program for profile and wavefront reduction. *Inter. Journal on Numerical Methods in Engineering*, **28**, 2651–2679, 1989.

S.W. Sloan and M.F. Randolph. Automatic element reordering for finite-element analysis with frontal schemes. *Inter. Journal on Numerical Methods in Engineering*, **19**, 1153–1181, 1983.

A Appendix: MC63 specification document

1 SUMMARY

This subroutine uses a variant of Sloan's algorithm to **generate an element assembly ordering** that is efficient when subsequently used with a frontal solver (for example, the packages MA42 and MA62). The number of floating-point operations and the storage required by a frontal solver for an unassembled finite-element matrix are dependent upon the order in which the elements are assembled; the variation in the performance of different element orderings can be significant. The assembly ordering obtained by MC63 is designed to reduce the maximum and root-mean-square (r.m.s.) wavefronts and the profile, which in turn reduce storage requirements and computation times for the frontal solver. Only the pattern of the finite elements is used.

Let n denote the number of variables (degrees of freedom) in the finite-element mesh. If f_i denotes the number of variables in the front before the ith elimination, the maximum wavefront is defined as

$$\max_{1\le i\le n}\{f_i\},$$

the profile is defined as

$$\sum_{i=1}^{n} f_i,$$

and the root-mean-square wavefront is defined as

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n} f_i^2}.$$

The user can choose between using a direct and an indirect element ordering algorithm. The indirect element ordering algorithm first orders the variables and then orders the elements. An option exists for working with supervariables in place of variables (a supervariable is a collection of one or more variables that belong to the same set of finite elements). If the problem has significantly fewer supervariables than variables, using supervariables will reduce the execution time of the ordering algorithm and will, in general, produce an ordering of comparable quality. Moreover, if the number of elements is less than the number of supervariables, it is generally quicker to order the elements using the direct ordering algorithm. However, for a given problem it is difficult to predict whether an indirect or a direct element ordering will produce the smaller wavefront.

ATTRIBUTES — Remark: Supersedes MC43. **Versions:** MC63A, MC63AD. **Calls:** MC60. **Date:** March 1998. **Origin:** J. A. Scott, Rutherford Appleton Laboratory. **Conditions on external use:** (i), (ii), (iii) and (iv).

2 HOW TO USE THE PACKAGE

2.1 Argument lists and calling sequence

There are three entries:

(a) MC63I/ID sets default values for control parameters. It should normally be called once prior to calling MC63A/AD.

(b) MC63A/AD reorders the elements.

(c) MC63B/BD computes, for a given element assembly order, the maximum front size, the profile, and the root-mean-square wavefront.

Only MC63A/AD provides extensive checks on the data. MC63B/BD optionally checks for duplicate and out-of-range entries. MC63B/BD is called by MC63A/AD but may also be used in combination with another algorithm for choosing an element assembly order.

2.1.1 To set default values for the control parameters

The single precision version

```
CALL MC63I(ICNTL)
```

The double precision version

```
CALL MC63ID(ICNTL)
```

`ICNTL` is an `INTEGER` array of length `10` that need not be set on entry. This array is used to hold control parameters. On return, `ICNTL` contains default values. If the user wishes to use values other than the defaults, the corresponding entries in `ICNTL` should be reset after the call to `MC63I/ID`. The control parameters are:

`ICNTL(1)` is the stream number for error messages and has the default value `6`. Printing of error messages is suppressed if `ICNTL(1)` ≤ 0.

`ICNTL(2)` is the stream number for warning messages and has the default value `6`. Printing of warning messages is suppressed if `ICNTL(2)` ≤ 0.

`ICNTL(3)` controls whether or not supervariables are used. If `ICNTL(3)` $= 0$, supervariables are used and if `ICNTL(3)` $= 1$, variables are used. The default value is `0`.

`ICNTL(4)` controls whether the user wishes to supply a global priority vector. If `ICNTL(4) = 0`, no priority vector is supplied and one is generated automatically using rooted level-set structures. If `ICNTL(4) = 1`, the user must supply a priority vector either in `ORDER` (`DIRECT = .TRUE.`) or in `PERM` (`DIRECT = .FALSE.`). The default value is `0`.

`ICNTL(5)` controls the action taken if duplicate and/or out-of-range indices are detected in the element variable lists. If `ICNTL(5)=0` and such indices are detected, they are removed, a warning is issued and the computation continues; if `ICNTL(5)=1`, the computation terminates. The default value is `0`.

`ICNTL(6)` controls whether the user wishes to supply the weights for the priority function. If `ICNTL(6)=0`, no weights are supplied and the weights used then depend on the algorithm chosen by the user (see Section 4). If `ICNTL(6) = 1`, the user must supply weights in `WT`. The default value is `0`.

`ICNTL(7)` to `ICNTL(10)` are currently not used but are given the default value `0`.

2.1.2 To reorder the elements

The single precision version

```
      CALL MC63A(DIRECT,N,NELT,NE,ELTVAR,ELTPTR,ORDER,PERM,NSUP,VARS,SVAR,
     +           WT,LIW,IW,LW,W,ICNTL,INFO,RINFO)
```

The double precision version

```
      CALL MC63AD(DIRECT,N,NELT,NE,ELTVAR,ELTPTR,ORDER,PERM,NSUP,VARS,SVAR,
     +           WT,LIW,IW,LW,W,ICNTL,INFO,RINFO)
```

`DIRECT` is a `LOGICAL` variable that must be set by the user. `DIRECT` controls which reordering algorithm is implemented. If `DIRECT = .TRUE.`, an direct element reordering algorithm is implemented; if `DIRECT = .FALSE.`, an indirect element reordering algorithm is implemented.

`N` is an `INTEGER` variable that must be set by the user to the largest integer used to index a variable in the finite-element problem. Note that the variables need not be numbered contiguously and, in this case, `N` will be larger than *n*, the number of degrees of freedom. This argument is not altered by the routine. **Restriction:** `N` ≥ 1.

`NELT` is an `INTEGER` variable that must be set by the user to the total number of finite elements in the problem. This argument is not altered by the routine. **Restriction:** `NELT` ≥ 1.

`NE` is an `INTEGER` variable that must be set by the user to be at least as large as the total number of entries in the element variable lists. This argument is not altered by the routine. **Restriction:** `NE` ≥ 1.

`ELTVAR` is an `INTEGER` array of length `NE`. On entry, `ELTVAR` must contain lists of the variable indices

belonging to each of the finite elements, with those for element 1 preceding those for element 2, and so on. If duplicate or variable indices outside the range [1,N] are detected, they are removed and the computation continues (ICNTL(5)=0, the default) or the computation terminates with ELTVAR unchanged (ICNTL(5)=1). On successful return, ELTVAR holds lists of supervariables (ICNTL(3)=0) or variables (ICNTL(3)=1) belonging to the elements.

ELTPTR is an INTEGER array of length NELT+1. On entry, ELTPTR(IELT) must contain the position in ELTVAR of the first variable in element IELT (IELT =1, 2,..., NELT), and ELTPTR(NELT+1) must be set to the position after the last variable in the last element. On return, ELTPTR holds corresponding data for the revised ELTVAR.

ORDER is an INTEGER array of length NELT. It need be set on entry only if DIRECT = .TRUE. **and** ICNTL(4) = 1. In this case, ORDER must hold positive global priority values for the elements (see Section 4, equation (2)). ORDER may be a permutation, in which case the element for which ORDER(IELT) = 1 is likely to be chosen first in the new assembly order and the element for which ORDER(IELT) = NELT is likely to be chosen last. On exit, the order in which the elements should be assembled is given by ORDER(1), ORDER(2),..., ORDER(NELT).

PERM is an INTEGER array that is only accessed if DIRECT = .FALSE. **and** ICNTL(4) = 1. In this case, PERM must be of length N and, if variable I is used to index a variable, PERM(I) must be set by the user to hold the positive global priority value for the variable I (see Section 4, equation (1)). PERM may be a permutation, in which case the variable for which PERM(I) = 1 is the variable with lowest priority and the variable for which PERM(I) = N is the variable with highest priority. This argument is not altered by the routine.

NSUP is an INTEGER variable that need not be set on entry. On successful return, if ICNTL(3)=0 (the default), NSUP holds the number of supervariables. If ICNTL(3)=1, on exit NSUP=N.

VARS is an INTEGER array of length N that need not be set on entry. On successful return, VARS(IS) holds the number of variables in supervariable IS, IS=1, 2,..., NSUP. If supervariables are not used, VARS(I) is set to 1 if I is used to index a variable and to 0 otherwise, I = 1, 2,..., N.

SVAR is an INTEGER array of length N that need not be set on entry. On successful return, SVAR(I) holds the supervariable to which variable I belongs, I = 1, 2,..., N. If variable I does not appear in the element lists, SVAR(I) = 0.

WT is a REAL (DOUBLE PRECISION in the D version) array of length 3. If ICNTL(6)=1, WT must be set by the user to hold the weights that are used in the priority function that is minimized when the element assembly order is computed (see Section 4). Default values are used if ICNTL(6)=0. WT(3) is not used if DIRECT = .FALSE. On return, WT holds the weights used in the priority function.

LIW is an INTEGER variable which defines the length of the work array IW. The workspace required depends upon whether the direct or the indirect element reordering algorithm is used, and upon whether supervariables are used (that is, the workspace depends upon the parameters DIRECT and ICNTL(3)). Lower bounds on the workspace needed are given by

LIW≥max(NE+3*NELT+NSUP+2, 2*N) if DIRECT = .TRUE. and ICNTL(3) = 0.

LIW≥ 2*(NELT+N+1)+max(NE, 4*NELT) if DIRECT = .TRUE. and ICNTL(3) = 1.

LIW≥ max(NE + NELT + 3*NSUP + 2, 2*N) if DIRECT = .FALSE. and ICNTL(3) = 0.

LIW≥3*N+2+NELT+max(NE, 3*N) if DIRECT = .FALSE. and ICNTL(3) = 1.

Upper bounds on the workspace required are given by

LIW≤max(NE, 4*NELT)+2*max(NELT,N)+3+NELT* (maxel+1) if DIRECT = .TRUE., where maxel is the maximum number of elements to which any one variable belongs.

LIW≤ max(NE, 3*N)+3*N+2+NELT*nodes*nodes if DIRECT = .FALSE., where nodes is the maximum number of variables in an element.

If LIW is too small, a value which will suffice is returned in INFO(5). This argument is not altered by the routine.

IW is an INTEGER array of length LIW. This array is used by the subroutine as workspace.

LW is an INTEGER variable which defines the length of the work array W. If DIRECT = .TRUE., LW must be at least NELT, and if DIRECT = .FALSE., LW must be at least NSUP. If LW is too small, a value which will suffice is returned in INFO(6). This argument is not altered by the routine.

W is a REAL (DOUBLE PRECISION in the D version) array of length LW. This array is used by the subroutine as workspace.

ICNTL is an INTEGER array of length 10 that must be set by the user to hold control parameters. Default values are set by a call to MC63I/ID. This argument is not altered by the routine.

INFO is an INTEGER array of dimension 15 that need not be set on entry.

INFO(1) is used as an error flag. On a successful exit, it is set to:

0 – Fully successful.

+1 – A warning has been issued. Further details are given in INFO(2), INFO(3), and RINFO.

If a fatal error has been detected, INFO(1) is set to a negative value:

–1 – N ≤ 0 or NELT ≤ 0 or NE < ELTPTR(NELT+1)-1. Immediate return with input parameters unchanged.

–2 – Failure due to insufficient space allocated to the array W. INFO(6) is set to a value that will suffice for LW.

–3 – ICNTL(5)=1 and duplicate or out-of-range indices detected in ELTVAR. The number of such indices is given by INFO(2) and INFO(3). The array ELTVAR is changed so that such indices are removed.

–4 – Failure due to insufficient space allocated to the array IW. INFO(5) is set to a value that will suffice for LIW.

INFO(2) holds the number of variable indices that were removed because they were duplicates.

INFO(3) holds the number of variable indices that were removed because they were out-of-range.

INFO(4) holds the number *n* of variables in the problem.

INFO(5) holds the amount of integer workspace used by the subroutine. If the user has provided insufficient integer workspace (INFO(1) = -4), INFO(5) is set to a value which will suffice for LIW (this value may be larger than the minimum workspace required).

INFO(6) holds the amount of real workspace used by the subroutine. If the user has provided insufficient real workspace (INFO(1) = -2), INFO(6) is set a value which will suffice for LW.

INFO(7) indicates whether the arrays ELTVAR and ELTPTR have been altered. If INFO(7) = 0, ELTVAR and ELTPTR are unchanged, otherwise they have been altered.

INFO(8) to INFO(15) are not currently used.

RINFO is a REAL (DOUBLE PRECISION in the D version) array of dimension 6 that need not be set on entry. On successful exit, RINFO contains the following information.

RINFO(1) holds the maximum wavefront for the initial element order 1, 2,..., NELT.

RINFO(2) holds the r.m.s. wavefront for the initial element order 1, 2,..., NELT.

RINFO(3) holds the profile for the initial element order 1, 2,..., NELT.

RINFO(4) holds the maximum wavefront for the permuted element order ORDER(1), ORDER(2),..., ORDER(NELT).

RINFO(5) holds the r.m.s. wavefront for the permuted element order ORDER(1), ORDER(2),..., ORDER(NELT).

RINFO(6) holds the profile for the permuted element order ORDER(1), ORDER(2),..., ORDER(NELT).

2.1.3 To compute statistics for a given element assembly order

The single precision version

```
CALL MC63B(JCNTL,N,NSUP,NELT,NE,ELTVAR,ELTPTR,VARS,ORDER,IW,INFO,RINFO)
```

The double precision version

```
CALL MC63BD(JCNTL,N,NSUP,NELT,NE,ELTVAR,ELTPTR,VARS,ORDER,IW,INFO,RINFO)
```

`JCNTL` is an `INTEGER` variable that controls whether the lists of variable indices are checked for errors.

- 0 – No checks made.
- +1 – Any duplicate or out-of-range indices are removed and the computation continues.
- –1 – The computation terminates if any duplicate or out-of-range indices are found.

If `JCNTL = +1` or `-1`, `INFO(2)` and `INFO(3)` provide information on the number of duplicate and out-of-range indices. This argument is not altered by the routine.

`N` is an `INTEGER` variable that must be set by the user to the largest integer used to index a variable in the finite-element problem. This argument is not altered by the routine.

`NSUP` is an `INTEGER` variable. If `MC63A/AD` has been called, it should be unchanged since the call to `MC63A/AD`. Otherwise, `NSUP` should be set to `N`, the largest integer used to index a variable in the finite-element problem. This argument is not altered by the routine.

`NELT` is an `INTEGER` variable that must be set by the user to the total number of finite elements in the problem. This argument is not altered by the routine.

`NE` is an `INTEGER` variable that must be set by the user to be at least as large as the total number of entries in the element variable lists. This argument is not altered by the routine.

`ELTVAR` is an `INTEGER` array of length `NE`. It may be as returned by `MC63A/AD`. Alternatively, it may be set by the user so that `ELTVAR` contains lists of the variable indices belonging to each of the finite elements, with those for element 1 preceding those for element 2, and so on. This argument is unchanged on exit unless `JCNTL = 1` and duplicate and/or out-of-range indices are detected.

`ELTPTR` is an `INTEGER` array of length `NELT+1`. It may be as returned by `MC63A/AD`. Alternatively, it may be set by the user so that `ELTPTR(I)` contains the position in `ELTVAR` of the first variable in element `I` (`I=1`, `2`,..., `NELT`), and `ELTPTR(NELT+1)` must be set to the position after the last variable in the last element. On exit, `ELTPTR` holds corresponding data for the revised `ELTVAR`.

`VARS` is an `INTEGER` array of length `NSUP`. If `MC63A/AD` was called, it should be unchanged since the call to `MC63A/AD` and it is not altered by the subroutine. Otherwise, (the case `NSUP` = `N`) `VARS` need not be set by the user.

`ORDER` is an `INTEGER` array of length `NELT` which must be set by the user so that the order in which the elements are assembled is given by `ORDER(1)`, `ORDER(2)`,..., `ORDER(NELT)`. This argument is not altered by the routine.

`IW` is an `INTEGER` array of length `NSUP`. This array is used by the subroutine as workspace.

`INFO` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length `4`.

`INFO(1)` is used as an error flag. On exit, it is set to:

- 0 – No duplicate or out-of-range indices detected.
- +1 – `JCNTL = 1` and some duplicate or out-of-range indices have been removed.
- –1 – `JCNTL = -1` and duplicate or out-of-range indices detected.

Further details are given in `INFO(2)` and `INFO(3)`.

`INFO(2)` holds the number of duplicate indices (only set if `JCNTL = 1`).

`INFO(3)` holds the number of out-of-range indices (only set if `JCNTL = 1`).

`INFO(4)` holds the number *n* of variables in the problem.

RINFO is a REAL (DOUBLE PRECISION in the D version) array of length 3. On exit, RINFO contains the following information.

RINFO(1) holds the maximum wavefront for the given element order.

RINFO(2) holds the r.m.s. wavefront for the given element order.

RINFO(3) holds the profile for the given element order.

3 GENERAL INFORMATION

Use of common: None.

Other routines called directly: Subroutines internal to the package are MC63C/CD, MC63D/DD, MC63E/ED, MC63F/FD, MC63G/GD. In addition, MC60C/CD, MC60H/HD, MC60L/LD, MC60O/OD are called.

Input/output: Error messages on unit ICNTL(1) (ICNTL(1) $\leq$ 0 suppresses them) and warnings on unit ICNTL(2) (ICNTL(2) $\leq$ 0 suppresses them).

Restrictions:

N $\geq$ 1, NELT $\geq$ 1, NE $\geq$ 1.

4 METHOD

MC63A/AD accepts lists of variables belonging to the elements and, after performing initial checks on the user's data, calls MC63B/BD to compute statistics for the natural element order 1,2,..,*nelt*. MC63B/BD also checks the element variable lists for out-of-range and duplicate indices. Any such entries are removed and the computation either continues after issuing a warning message or terminates if this has been requested.

If supervariables are wanted (ICNTL(3) = 0), they are constructed using subroutine MC60O/OD from the Harwell Subroutine Library profile reduction package MC60. Otherwise, each variable is treated as a supervariable. The element variable lists are overwritten by element supervariable lists. A map of variable to supervariable indices allows the user to later restore the element variable lists, if desired.

For each supervariable, the number of elements involving it is counted. Lists of the elements associated with the supervariables are then constructed. If the user has selected the direct element reordering algorithm (DIRECT = .TRUE), the element connectivity graph is constructed from the supervariable lists, otherwise the supervariable connectivity graph is constructed from the element lists.

Both the direct and indirect algorithms are based upon the modifications of Sloan's algorithm (Sloan 1986) first introduced by Duff, Reid, and Scott (1989).

In the indirect element reordering algorithm (DIRECT = .FALSE), MC60C/CD is used to reorder the supervariables. The priority function that is minimized when choosing the next supervariable in the order is

$$\mathtt{WT(1)}\ deg(s) + \mathtt{WT(2)}\ v\, glob(s) \qquad (1)$$

where $deg(s)$ is the number of variables that will enter the front if supervariable s is chosen next, v is a normalizing factor, and $glob(s)$ is the (positive) global priority value of supervariable s (generated automatically or provided in PERM). On the basis of our numerical experiments, the default values for the weights WT(1), WT(2) are (2,1) if the global priority vector is based on a rooted level-set structure (ICNTL(4) = 0) and (1,2) for a global priority vector based on the spectral method. Full details are given in Reid and Scott (1998). Once the supervariables have been reordered, the elements are resequenced in ascending order of their earliest supervariable in the new supervariable order. The new supervariable indices are not preserved.

In the direct element ordering algorithm, the element connectivity graph is relabeled using the priority function

$$\mathtt{WT(1)}\ ngain(ielt) + \mathtt{WT(2)}\ v\, glob(ielt) + \mathtt{WT(3)}\ nadj(ielt) \qquad (2)$$

where $ngain(ielt)$ is the number of variables element *ielt* will introduce into the front less the number that can then be eliminated, $nadj(ielt)$ is the number of elements adjacent to element *ielt* that have not yet been relabeled, v is a normalizing factor, and $glob(ielt)$ is the (positive) global priority value of element *ielt* (generated

automatically or provided in ORDER). At each stage, from a list of eligible elements, the element that minimizes the priority function is chosen to be next in the element assembly order. The default values for the weights are (10,5,1) if the global priority vector is based on a rooted level-set structure (ICNTL(4) = 0) and (1,2,0) for a global priority vector based on the spectral method. For full details the user is referred to Scott (1998).

A final call to MC63B/BD (without error checking) computes statistics for the new element order.

References

Duff, I. S., Reid, J. K., and Scott, J. A. (1989). The use of profile reduction algorithms with a frontal code. *Int. J. Numer. Meth. Engng.*

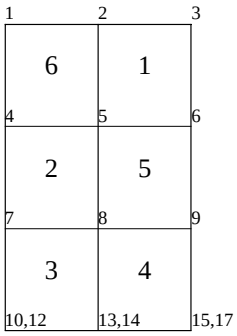
Reid, J. K. and Scott, J. A. (1998). Ordering symmetric sparse matrices for small profile and wavefront. Technical Report RAL-TR-98-016, Rutherford Appleton Laboratory.

Scott, J. A. (1998). On ordering elements for a frontal solver. Technical Report RAL-TR-98-031, Rutherford Appleton Laboratory.

Sloan, S. W. (1986). An algorithm for profile and wavefront reduction of sparse matrices. Inter. J. Numer. Meth. Engng. **23**, 239-251.

5 EXAMPLE OF USE

The following program provides an example of the use of MC63. We wish to reorder the elements in the following simple finite-element mesh comprising six 4-noded quadrilateral elements. The elements are initially numbered arbitrarily. Note that the variables are not numbered contiguously and at some nodes there is more than one freedom.




```
C  Example to illustrate the use of MC63A.
C  Both the direct and the indirect element reordering algorithms
C  are employed.

C     .. Parameters ..
      INTEGER MELT,MXN,MZ,LIW
      PARAMETER (MELT=6,MXN=20,MZ=30,LIW=200)
C     ..
C     .. Local Scalars ..
      INTEGER I,IDUM,LW,N,NE,NELT,NSUP
      LOGICAL DIRECT
C     ..
C     .. Local Arrays ..
      REAL WT(3),RINFO(6),W(MXN)
      INTEGER CPTR(MELT+1),CVAR(MZ),ELTPTR(MELT+1),ELTVAR(MZ),
     +        ICNTL(10),INFO(15),IW(LIW),
     +        ORDER(MELT),PERM(1),SVAR(MXN),VARS(MXN)
C     ..
C     .. External Subroutines ..
      EXTERNAL MC63A,MC63I
C     ..
C   Read in the finite-element data
      READ (5,*) N,NELT
```

```
      READ (5,*) (ELTPTR(I),I=1,NELT+1)
      NE = ELTPTR(NELT+1) - 1
      READ (5,*) (ELTVAR(I),I=1,NE)
      LW = MXN

C Take a copy of ELTVAR, ELTPTR as altered by MC63A/AD
C and we want to run direct and indirect algorithms
      DO 5 I = 1,NELT+1
         CPTR(I) = ELTPTR(I)
    5 CONTINUE
      DO 10 I = 1,NE
         CVAR(I) = ELTVAR(I)
   10 CONTINUE

      CALL MC63I(ICNTL)

      DO 20 IDUM = 1,2

         IF (IDUM.EQ.1) THEN
            DIRECT = .TRUE.
            WRITE (6,'(/3X,A)') '*** Direct element reordering ***'
         ELSE
            DIRECT = .FALSE.
            WRITE (6,'(/3X,A)') '*** Indirect element reordering ***'
C Reset ELTVAR, ELTPTR
            DO 15 I = 1,NELT+1
               ELTPTR(I) = CPTR(I)
   15       CONTINUE
            DO 16 I = 1,NE
               ELTVAR(I) = CVAR(I)
   16       CONTINUE
         END IF

         CALL MC63A(DIRECT,N,NELT,NE,ELTVAR,ELTPTR,ORDER,PERM,NSUP,
     +              VARS,SVAR,WT,LIW,IW,LW,W,ICNTL,INFO,RINFO)

C  Check for errors
         IF (INFO(1).LT.0) GO TO 30

         WRITE (6,'(A,I12,I12/A,1P,D12.4,1P,D12.4/
     +       A,1P,D12.4,1P,D12.4)')
     +     ' Original/new max. wavefront             = ',
     +          INFO(RINFO(1)),INT(RINFO(4)),
     +     ' Original/new r.m.s. wavefront           = ',
     +          RINFO(2),RINFO(5),
     +     ' Original/new profile                    = ',
     +          RINFO(3),RINFO(6)
         WRITE (6,'(A,I12/A,I12,I12)')
     +     ' Workspace used                          = ',
     +          INFO(5),
     +     ' Number of variables/supervariables      = ',
     +          INFO(4),NSUP
         WRITE (6,'(/3X,A/6I5)') 'The new element order is :',
     +     (ORDER(I),I=1,NELT)

   20 CONTINUE
      GO TO 40

   30 WRITE (6,*) ' Unexpected error return'

   40 STOP
      END
```

The input data used for this problem is:

```
  17   6
   1   5   9  15  21  25  29
   2   5   3   6   4   5   7   8   7   8  10  12  4  13
   8  13   9  14  17  15   5   8   9   6   1   2   5   4
```

This produces the following output:

```
   *** Direct element reordering ***
 Original/new max. wavefront             =           10           7
 Original/new r.m.s. wavefront           =   6.3823D+00  4.6476D+00
 Original/new profile                    =   8.7000D+01  6.6000D+01
 Workspace used                          =           86
 Number of variables/supervariables      =           15          13

   The new element order is :
    1    6    5    2    3    4

   *** Indirect element reordering ***
 Original/new max. wavefront             =           10           7
 Original/new r.m.s. wavefront           =   6.3823D+00  4.6476D+00
 Original/new profile                    =   8.7000D+01  6.6000D+01
 Workspace used                          =          142
 Number of variables/supervariables      =           15          13

   The new element order is :
    1    6    5    2    3    4
```