INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng. **45**, 1737–1755 (1999)

ORDERING SYMMETRIC SPARSE MATRICES FOR SMALL PROFILE AND WAVEFRONT

J. K. REID* AND J. A. SCOTT

Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Didcot OX11 0QX, U.K.

SUMMARY

The ordering of large sparse symmetric matrices for small profile and wavefront or for small bandwidth is important for the efficiency of frontal and variable-band solvers. In this paper, we look at the computation of pseudoperipheral nodes and compare the effectiveness of using an algorithm based on level-set structures with using the spectral method as the basis of the Reverse Cuthill–McKee algorithm for bandwidth reduction. We also consider a number of ways of improving the performance and efficiency of Sloan's algorithm for profile and wavefront reduction, including the use of different weights, the use of supervariables, and implementing the priority queue as a binary heap. We also examine the use of the spectral ordering in combination with Sloan's algorithm. The design of software to implement the reverse Cuthill–McKee algorithm and a modified Sloan's algorithm is discussed. Extensive numerical experiments that justify our choice of algorithm are reported on. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: sparse matrices; symmetric pattern; profile reduction; Sloan algorithm; reverse Cuthill–McKee algorithm; spectral method

1. INTRODUCTION

We consider the ordering of symmetric sparse matrices for small profile and wavefront or for small bandwidth. We are primarily concerned with matrices that are positive definite, so we work only with the pattern of the matrix and do not take into account any permutations needed for numerical stability. The work is useful for a matrix that is non-definite or is symmetric only in the pattern of its entries, but in these cases it must be appreciated that the actual factorization may be more expensive and require more storage. For finite-element applications, we assume that the matrix has been assembled. We treat unassembled finite-element matrices differently and this will not be considered here.

In recent years, much attention has been paid to the problem of ordering symmetric sparse systems (see, for example [1] for a discussion and list of references). One method which has been widely used for profile reduction is that of Sloan [2, 3]. Sloan exploits the close relationship between a symmetric matrix $\mathbf{A} = \{a_{ij}\}$ of order n 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 non-zero. Sloan's algorithm has two distinct phases. In the first, a start node and an end node are chosen. In the

* Correspondence to: J. K. Reid, Computational Science and Engineering Department, Rutherford Appleton Laboratory, Atlas Centre, Chilton, Didcot OX11 0QX, U.K.

CCC 0029-5981/99/241737–19$17.50

Received 9 February 1998
Revised 16 November 1998

second phase, the chosen start node is numbered first and a list of nodes that are eligible to be numbered next is formed. At each stage of the numbering, the list of eligible nodes comprises the neighbours of nodes which have already been numbered and their neighbours. The next node to be numbered is selected from the list of eligible nodes by means of a priority function. A node has a high priority if it causes either no increase or only a small increase to the current front size and is at a large distance from the end node.

The Harwell Subroutine Library code MC40 [4] implements the ordering algorithm of Sloan [2] and has been in satisfactory use for a decade. We decided that a revision was needed mainly because Kumfert and Pothen [5] have found that, for the larger problems that are handled nowadays, there is a considerable efficiency gain from the use of a binary heap to manage the list of eligible nodes in the second phase of Sloan's algorithm. Another reason is for the economy of working with supervariables (sets of variables for which the corresponding matrix columns have identical patterns) when the number of supervariables is significantly less than the number of variables. We have added an option that permits users to provide a global priority vector because Kumfert and Pothen have found that the final ordering can be significantly better if we use a hybrid algorithm that combines a spectral ordering (see, for example [6]) with the Sloan algorithm. We have also taken the opportunity to revise the code in a number of other ways, including adding an option for performing the Reverse Cuthill–McKee algorithm and allowing the user to specify the weights of Sloan's algorithm. The new code is called MC60, and we also provide a simple driver called MC61.

This paper is organized as follows. In Section 2, we briefly review frontal and variable-band methods. In Section 3, we look at computing start and end nodes for Sloan's algorithm. We examine modifications to improve the performance of the algorithm of Gibbs *et al.* [7] for finding a pseudodiameter. We also look at using the spectral method to find a pseudodiameter and consider the effect of interchanging the ends of the pseudodiameter on the quality of the Reverse Cuthill–McKee algorithm. The numbering phase of Sloan's algorithm is considered in Section 4. We discuss the priority function and compare the performance of a simple sequential search for finding the node of highest priority with that of a binary heap implementation. We look at the effect of adjusting the weights in the priority function and of using the spectral pseudodiameter for the start and end nodes. In Section 5, we describe a modified version of the hybrid method of Kumfert and Pothen [5]. The design of our codes MC60 and MC61 is discussed in Section 6. Finally, a concluding discussion is given in Section 7.

To illustrate our ideas and findings, throughout this paper we use the test examples of Everstine [8] and of Kumfert and Pothen [5]. It should be noted that, although the Everstine problems have been widely used for testing algorithms of this kind, they are small by current standards. Their order varies from 59 to 2680 and Sloan reports root-mean-square wavefronts, following his ordering, varying from 3 to 40. The orders for the Kumfert and Pothen set vary from 6019 to 100 196 and the root-mean-square wavefronts, following Sloan's ordering, vary from 59 to 1399.

All the results presented in this paper are for Fortran 77 code compiled with the Edinburgh Portable Compilers, Ltd (EPC) Fortran 90 compiler with optimization – O running on a 143 MHz Sun Ultra 1. All timings are in CPU seconds.

2. BACKGROUND

Two methods for solving large sparse symmetric systems of equations $\mathbf{Ax} = \mathbf{b}$ that are widely used, especially in finite-element analysis, are the variable-band (profile) and frontal methods. The

efficiency of these methods is affected substantially by the ordering of the variables. In this section, we briefly discuss the parameters which measure the efficiency of variable-band and frontal methods.

2.1. Frontal and variable-band methods

Sloan's method aims to find an elimination order that is suitable for the frontal method applied to a symmetric and positive-definite matrix. A variable is in the front if it has not yet been eliminated but is adjacent to a variable that has been eliminated or is about to be eliminated. There is an underlying assumption that full matrix storage is used for the frontal matrix (the submatrix corresponding to the rows and columns of the front) and that no advantage is taken of zeros within it. The order of the frontal matrix is known as the *wavefront*. Of interest is

(1) the maximum wavefront, since this affects the in-core storage needed,
(2) the sum of the wavefronts, known as the *profile*, since this is the total storage needed for either of the factors, and
(3) the root-mean-square wavefront, since the work performed when eliminating a variable is proportional to the square of the current wavefront.

The elimination order is also relevant for the variable-band method. Here, instead of the maximum wavefront, of relevance is

(1) the maximum semibandwidth, which affects the number of matrix rows that need to be held in-core at once (unless we are willing to reread parts of the factors from disk).

If no advantage is taken of zeros within the band, the profile is of relevance because again it is the total storage needed for either of the factors. The root-mean-square wavefront is also important since it bears the same relationship to the work performed as for the frontal method.

2.2. The row-by-row frontal method for unsymmetric matrices

In the frontal method, the matrix **A** need not be assembled explicitly. Instead, the assembly and elimination operations are interleaved with each variable being eliminated as soon as its row and column are fully summed. If the matrix is already assembled, a frontal code may accept the matrix row by row and treat each row as an element matrix (see, for example, [9] Section 10.6). We will call this the *row-by-row* frontal method. The Harwell Subroutine Library frontal code MA42 [10] includes such an option. In the row-by-row frontal method, a variable is eliminated when its index appears in an index list for the entries of a row for the last time. We use the terms *row front size* and *column front size* for the numbers of rows and columns in the rectangular frontal matrix involved. For efficiency, the rows need to be numbered for small row and column front sizes. Of interest here are

(1) the maximum row and column front sizes, and
(2) the root-mean-square row and column front sizes

For the permuted matrix, if the matrix **A** is unsymmetric but has a symmetric sparsity pattern, Sloan's method may be used to give an efficient elimination order. We can then obtain a row ordering for the row-by-row frontal method by first ordering all the rows that have an entry in the

column of the first variable in the elimination order, then any remaining rows that have an entry in the column of the second variable, and so on.

Given this row order, a row-by-row frontal code will use a very similar elimination order to that specified, but it is unlikely to be identical since several columns may become fully summed at the same time. Usually, the sequence of numbers of rows in the front will be identical to the sequence of wavefront sizes, but even this need not be so. It may happen that a column becomes fully summed before its variable is reached in the specified elimination order; in this case, it can be eliminated at once the numbers of rows in the front will be less than the corresponding wavefront sizes until the variable's position in the specified elimination order is reached. For example, consider the matrix with entries

$$\begin{array}{cccc} \times & \times & \times & \\ \times & \times & & \times \\ \times & & \times & \\ & \times & & \times \end{array}$$

and the natural elimination order 1, 2, 3, 4. Rows 1–3 are loaded into the front, then variable 1 is eliminated. At this point, variable 3 is fully summed and can be eliminated although it has not yet been reached in the elimination order.

3. FINDING START AND END NODES FOR SLOAN'S METHOD

In this section, we consider finding pairs of nodes that are at maximum or nearly maximum distance apart, since experience has shown that such nodes are good candidates for starting nodes for profile and wavefront reduction algorithms and for bandwidth reduction algorithms (see, for example, [2, 7, 11, 12]). In our discussion we assume that the matrix $\mathbf{A}$ is irreducible so that its associated graph G is connected (if not, we work with each component of the graph separately).

We first introduce notation that we will use throughout this paper and recall some standard terminology and concepts from elementary graph theory (see [7]). The *degree* of node $P \in G$ is the number of nodes that are adjacent to P. The *distance* $d(P, Q)$ between two nodes P and Q in G is defined to be the length of the shortest path connecting them (one less than the number of nodes on the path). The *diameter length* of G is the greatest distance between any two nodes in G. A *diameter* is a shortest path between two nodes P and Q whose distance apart is equal to the diameter length. A *pseudodiameter* is either a diameter or a shortest path between two nodes in G whose distance apart is slightly less than the diameter length. An end of a pseudodiameter is called a *pseudoperipheral node*. It is convenient here to refer to pseudodiameters, but actually we will only ever be interested in the pairs of pseudoperipheral nodes that define them.

3.1. Finding a pseudodiameter using level sets

Gibbs *et al.* [7] find a pseudodiameter by constructing level-set structures. The algorithm we propose is a modified version of their procedure. The *level-set structure* rooted at a node P is defined as the partitioning of the nodes in G into level sets $L_1(P), L_2(P), \ldots, L_h(P)$ such that

(i) $L_1(P) = \{P\}$ and

(ii) for $i > 1$, $L_i(P)$ is the set of all nodes that are adjacent to nodes in $L_{i-1}(P)$ but are not in $L_1(P), L_2(P), \ldots, L_{i-1}(P)$.

Note that all the nodes in $L_{i+1}(P)$ are at the distance i from P. The level-set structure rooted at P is denoted by $\mathbf{L}(P)$. We refer to the number h of level sets in a level-set structure as its *depth* and the greatest number of nodes in a level set as its *width*. Gibbs *et al.*, choose a starting node P of minimum degree and generate $\mathbf{L}(P)$. They then generate the level structures rooted at each of the nodes in the final level set $L_h(P)$. If the level-set structure $\mathbf{L}(Q)$ rooted at such a node Q has a greater depth than $\mathbf{L}(P)$, the whole process is recommenced with Q replacing P.

Constructing level-set structures for all the nodes in the final level set is obviously expensive. George and Liu [13] therefore recommended terminating the construction of any level-set structure whose width reaches or exceeds that of the narrowest level-set structure so far found. The significance of the width is that it is closely related to the wavefront of the matrix if the level-set structure is used for ordering. Lewis [14] recommended that the nodes should also be sorted by degree, since pseudoperipheral nodes usually have low degree. Sloan [2] incorporated both these modifications into his algorithm for finding pseudoperipheral nodes. He also used the empirical observation that nodes with high degrees are not often selected as potential start or end nodes to introduce a shrinking strategy that reduces the number of nodes in the final level set L_h for which level-set structures are generated. Sloan chose to shrink L_h by taking the first $\mathrm{int}(m/2) + 1$ nodes (sorted in ascending sequence of degree), where int is the Fortran int function (truncation towards zero) and m is the number of nodes in L_h.

In earlier work [4], we tried the strategy of rejecting any node in L_h that had a neighbour that had already been tested, but rejected this on the grounds of its being more expensive than Sloan's shrinking strategy. Instead, we decided to limit the search to one representative of each degree, which we found to be significantly more economical while having little effect on the quality of the final ordering. This strategy was used in the code MC40. When publishing a Fortran implementation of his algorithm, Sloan ([3, p. 2655]) followed us, saying 'this often minimizes the number of level structures that need to be generated without affecting the quality of the pseudoperipheral nodes'.

Since two nodes may have the same degree while being well separated with quite different level-set structures the strategy that we now recommend is to consider up to five nodes in the final level set in order of increasing degree, omitting any that is a neighbour of a node already considered. We follow George and Liu in terminating the construction of any level-set structure whose width reaches or exceeds that of the narrowest level-set structure so far found. We will refer to our procedure as the MGPS (modified Gibbs Poole Stockmeyer) algorithm. On the Kumfert and Pothen test matrices, we found no case where the depth was increased by using this strategy in place of that proposed of Duff *et al.* [4], but for a few (notably *nasasarb* and *onera_dual*) the width was significantly reduced. The computation times were generally very similar. We present in Table I the cases that showed different widths. Column 3 shows the number of nodes of the final level set that were considered.

Cuthill and McKee [15] proposed that the ordering associated with the level-set structure be used as a basis for an ordering for the variable-band method and George [16] found that there are advantages in reversing the resulting order. For an explanation of why this is so, see [9, p. 155]. We provide this ordering as an option in our new codes MC60 and MC61 (see Section 6). In Table I, we also present the resulting semibandwidths when the level-set structures

Table I. The cases where the new pseudodiameter algorithm gave different widths

Problem	Code	No. tries	Time	Level-set		RCM semi-bandwidth
				Depth	Width	
barth5	MC40	1	0·06	103	380	394
	MGPS	5	0·13	103	359	373
copter2	MC40	2	0·45	54	2226	2322
	MGPS	2	0·45	54	2204	2280
nasasarb	MC40	3	0·92	176	864	882
	MGPS	5	1·16	176	540	577
onera_dual	MC40	1	0·58	84	3270	3479
	MGPS	2	0·58	84	2712	2768
shuttle_eddy	MC40	4	0·07	176	236	239
	MGPS	5	0·07	176	225	227
tandem_vtx	MC40	7	0·28	30	1471	1602
	MGPS	5	0·25	30	1472	1565

computed by our new strategy and by the MC40 strategy are used for the Reverse Cuthill–McKee ordering. We refer to this as the RCM ordering.

We remark that our limit of five nodes in the final level set is somewhat arbitrary, but without such a limit, we found that we could do significantly more work without improving the quality of the final result.

3.2. *Interchanging the ends of the pseudodiameter*

In almost all of our test problems, we found that the widths seen from the two ends of the pseudodiameter were different, sometimes by a significant amount. The columns labelled Widths MGPS in Table II show the widths from the opposite ends. If the width is important, there seems to be no alternative to computing it from both ends and choosing as the start node the one whose level-set structure has the lesser width. We do this in our codes MC60 and MC61. This usually involves no overhead, but can require one more level-set structure to be constructed to find the distances to the end node (we need to do this anyway whenever the most recently computed level-set structure is not of least width, see Section 6.3).

3.3. *The spectral pseudodiameter*

In this section, we briefly discuss a recent method that has been proposed for finding a pseudodiameter, without constructing level-set structures. Barnard *et al.* [6] and Paulino *et al.* [17] described a spectral algorithm that associates a Laplacian matrix **L** with the given matrix **A** with a symmetric sparsity pattern,

$$\mathbf{L} = \{l_{ij}\} = \begin{cases} -1, & i \neq j, \quad a_{ij} \neq 0 \\ 0, & i \neq j, \quad a_{ij} = 0 \\ \sum_{k \neq i} |l_{ik}|, & i = j \end{cases}$$

Table II. Depths, widths, and RCM semibandwidths, using the MGPS and spectral orderings from both ends of the pseudodiameter

	Depths			Widths				RCM semibandwidths			
Problem	MGPS	Spectral		MGPS		Spectral		MGPS		Spectral	
barth	**70**	64	69	192	**180**	190	192	200	**194**	201	199
barth4	**71**	65	66	212	196	**184**	202	220	198	**188**	208
barth5	**103**	102	102	**359**	392	399	377	**373**	403	412	389
bcsstk30	**34**	32	29	2504	2639	2639	**2504**	2827	**2814**	**2814**	2827
commanche_dual	**157**	140	156	152	127	**123**	127	158	134	**129**	134
copter1	**42**	**42**	**42**	917	**915**	**915**	917	**935**	963	963	**935**
copter2	**54**	53	53	2204	2609	2869	**2197**	2280	2754	2950	**2270**
finance256	**56**	53	52	**2010**	**2010**	**2010**	**2010**	2016	2016	**2014**	**2014**
finance512	**88**	87	87	**1208**	1211	1211	1211	**1307**	**1307**	1319	1319
ford1	**143**	130	134	**252**	303	319	295	**257**	312	332	307
ford2	**244**	234	230	956	961	1009	**939**	962	986	1021	**955**
nassasrb	**176**	161	173	**540**	864	1068	900	**577**	881	1080	944
onera_dual	**84**	80	72	**2712**	5074	3780	3454	**2768**	5164	3809	3535
pds10	**16**	**16**	**16**	3419	3290	3419	**2917**	4117	3803	4117	**3392**
shuttle_eddy	**176**	170	**176**	225	**167**	198	**167**	227	**177**	201	**177**
skirt	**57**	**57**	56	1879	**1776**	1913	1972	2071	**1994**	2071	2174
tandem_dual	103	**104**	98	**2139**	2331	2285	2167	2206	2387	2325	**2173**
tandem_vtx	30	**31**	**31**	**1472**	1774	1584	1508	**1565**	1848	1681	1571

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 non-increasing or non-decreasing order and Barnard *et al.* found that this gave good profile sizes.

Paulino *et al.* [1] suggested using the first and last nodes of the spectral permutation to define a pseudodiameter. They take the better of the RCM orderings based on these two nodes. In Table II, we show the depths, widths, and RCM semibandwidths that result from using the two ends of the MGPS and spectral pseudodiameters. In our experiments, the Fiedler vector was obtained using Chaco 2·0 [18]. We used the SymmLQ/RQI option and the input parameters were chosen to be the same as those used by Kumfert and Pothen [5].

We remark that for the spectral pseudodiameter, if the level-set structure rooted at one end is constructed, the other end does not necessarily lie in the final level set. Therefore, the depths as well as the widths of the level-set structures rooted at each end of the pseudodiameter can differ. We see from our results that only for *tandem_vtx* and one end of *tandem_dual* does the spectral pseudodiameter yield a level-set structure with a greater depth than the MGPS pseudodiameter, but in about half the cases, it produces a narrower width and smaller RCM semibandwidth. We highlight in bold the greatest depths, the narrowest widths, and the smallest RCM semibandwidths. For both methods, the importance of using the end of the pseudodiameter with the narrower level-set structure is apparent. Paulino *et al.* report the results of taking the better of the two ends for the spectral method, but do not try reversing the ends for the Gibbs Poole Stockmeyer algorithm. It is our belief that it is because of this that in their paper the spectral method yielded slightly improved results and not because the spectral pseudodiameter is

inherently superior for bandwidth reduction algorithms. In our code, we always use the end of the pseudodiameter that yields the level-set structure with the lesser width.

3.4. Using the supervariable graph

For efficiency, our new codes offer the option of working with supervariables. A *supervariable* is defined to be the set of variables that correspond to a set of columns of **A** with identical patterns. A permutation is constructed that places the variables of each supervariable together. The pattern of **A** is replaced by that of the permuted matrix represented as supervariables (that is, by its condensed or compressed equivalent). We call this the *supervariable graph* and is used in place of the graph associated with the original matrix. The potential savings in the computation times by using the supervariable graph are illustrated by Duff *et al.* (see also Table VIII in Section 7).

When a node is introduced into a level set in the original graph, all the nodes of its supervariable will be introduced too, unless one of them is the start node. The other nodes of the start supervariable will be in level 2 if the corresponding matrix rows have diagonal entries and in level 3 otherwise. Thus there is no significant difference between the properties of the variable and supervariable graphs. The depths will be the same, except for the trivial cases where the depth is 2 or 3.

In processing the supervariable graph, we take the numbers of variables in the supervariables into account when calculating the width of a level-set structure, but not for the degrees of the supervariables in the list of potential start nodes. Our reasoning for these choices is that

(1) the width has a direct bearing on the wavefront or semibandwidth when the ordering is used without alteration and calculating it is a small overhead in the loop that adds supervariables to the level set; and
(2) the supervariable degree is likely to have greater topological relevance and, if there are substantially fewer supervariables than variables, is significantly cheaper to calculate.

4. SLOAN'S ALGORITHM

In this section, we discuss the second phase of Sloan's algorithm, that is, the numbering phase. We will again assume that the matrix **A** is irreducible. It is straightforward to apply the algorithm to each component of a reducible (block diagonal) matrix and Sloan's code (and ours) allows for this.

4.1. The Sloan priority function

In the first phase of his algorithm, Sloan finds a pseudodiameter and, in the second phase, uses this to guide his ordering. One end s of the pseudodiameter is used as the start node and the other e is used as a target end node. In fact, Sloan ensures that the position of a variable in his ordering is not very far away 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. He begins at the start node s and uses the priority function

$$P_i = -W_1 c_i + W_2 d(i, e) \tag{1}$$

for node i, where W_1 and W_2 are integer weights, c_i (which he calls 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. 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). Based on his numerical experiments, Sloan recommends the pair (2, 1) for the weights. Following further experiments, we also used these values in our Harwell Subroutine Library code MC40.

If the current degree c_i of a node drops to zero, it should be chosen as the next node to be renumbered since eliminating the corresponding variable next is bound to reduce the size of the front. Since testing for zero c_i is not a big overhead, we do this in our code. This can only improve the quality of the result, but such a node is likely to be chosen anyway if the ratio W_1/W_2 is large because c_i cannot be negative.

4.2. Sloan's search

For his list of eligible nodes, Sloan takes all nodes that are in the front (neighbours of one or more renumbered nodes) or are neighbours of one or more nodes in the front. He performs a simple sequential search of the list to find the node with highest priority that is to be numbered next. Sloan noted that the simple search was faster than using a binary heap search for most of Everstine's test problems [8], but suggested that the binary heap search will inevitably become the method of choice for large problems where the root-mean-square wavefront exceeds several hundred nodes. The recent work of Kumfert and Pothen [5] confirms this expectation. To make our code efficient both on the small problems used by Sloan and the much larger problems which are common today, we commence with code that performs Sloan's simple search, but switch to code that uses a binary heap if the number of eligible nodes exceeds a threshold. Our experience is that the performance is not very sensitive to this threshold. Based on our numerical experiments, we use a threshold of 100 in our code. We show timings for six test problems in Table III, chosen to illustrate the performance in cases that each vary from the next by a factor of about 10 in the time taken when a simple search is used (threshold n). For small problems, our method with threshold 100 can be slightly less efficient than the simple search (probably because once we have switched to the heap, we do not return to the simple search), but there are substantial gains on the largest cases.

4.3. Managing the binary heap

We hold the list of indices of eligible nodes in any array QUEUE, with the root in QUEUE (1), its children in QUEUE (2) and QUEUE (3) , the children of QUEUE (2) in QUEUE (4) and QUEUE (5) , etc. Thus the parent of QUEUE (J) is always in QUEUE (J/2) . We ensure that the priority value for a node is never more than that of its parent. As a result, the root is always the node with the highest priority, so no search is needed to choose the next node for renumbering.

To restore the binary heap after removing the root, we move its child with greater priority into its place, then do the same for the child, continuing until the bottom of the heap is reached. About $\log_2 l$ steps are needed for a list of length l.

When a node is added to the list of eligible nodes, it is added to the bottom of the heap. To ensure that the heap still has the required properties, we need to compare the value of the priority

Table III. Effect on reordering times of the threshold for using the heap

Order (n)	DWT 59	DWT 221	DWT869	*shuttle_eddy*	*tandem_vtx*	*onera_dual*
Threshold	59	221	869	10 429	18 454	85 567
0	0·00042	0·0028	0·015	0·16	0·53	1·7
10	0·00041	0·0028	0·015	0·16	0·53	1·7
20	0·00035	0·0028	0·016	0·16	0·53	1·7
50	0·00035	0·0026	0·015	0·16	0·53	1·7
100		0·0025	0·014	0·16	0·53	1·7
500			0·014	0·19	0·54	1·8
1000				0·19	0·64	1·8
n	0·00035	0·0024	0·014	0·19	1·51	25·4

function with that of its parent. If necessary, an interchange with the parent is made and the same comparison is made at the parent node. This continues until the root is reached or a correctly ordered parent and child is reached. At most $\log_2 l$ steps are needed for a list of length l.

The part of the algorithm which is potentially expensive is maintaining the priorities of the eligible nodes as nodes are renumbered. When the value of the priority function of an eligible node changes, it is always an increase caused by a neighbour being included in the front. We need to compare the new value of the priority function with that of its parent as in the previous paragraph. At most $\log_2 l$ steps are again needed, but our experience is that in most cases, no interchanges at all are needed.

4.4. Interchanging the start and end nodes

We observed in Table I that which end of the pseudodiameter is chosen as the start node can have a marked effect on the width of the level-set structure and on the RCM semibandwidth. To a lesser extent, which end is chosen as the start node affects the Sloan algorithm. In Table IV, we show the level-set widths and the Sloan profiles for the Kumfert and Pothen test set. For each problem, we report the results for the better of the pairs of weights (2, 1) and (16, 1) (see Section 4.5 for a discussion of the choice of weights). In column 2, we give the narrowest width and in column 4, the corresponding profile. We highlight the best profile (if the profiles for both ends of the pseudodiameter differ by less than 2 per cent, both are highlighted). It can be seen that for most problems the final profile is not very sensitive to which end is used as the start node but there appears to be a slight advantage in choosing the pseudoperipheral node that gives the narrowest width as the start node. We therefore take this node as the start node s in our code, since we feel that the added expense of running the second phase of Sloan's algorithm using both nodes would not be justified.

4.5. Adjusting Sloan's weights

As already mentioned, Sloan recommends the pair (2, 1) for the weights. However, the results of Kumfert and Pothen [5] indicate that, for some problems, there are considerable advantages in using other values. We have examined the profiles for the 13 pairs of weights (1, 64), (1, 32), (1, 16), . . . , (1, 1), (2, 1), . . . , (64, 1) on all the Everstine and Kumfert and Pothen test matrices.

Table IV. MGPS widths and Sloan profiles (in millions) for the two pseudoperipheral nodes

Problem	MGPS widths		Sloan profiles	
barth	180	192	**0·47**	**0·47**
barth4	196	212	**0·33**	**0·34**
barth5	359	392	**1·44**	1·49
bcsstk30	1208	1211	16·15	**11·16**
commanche_dual	127	152	**0·33**	**0·33**
copter1	915	917	**6·05**	**6·05**
copter2	2204	2609	**37·96**	38·87
finance256	2012	2012	**6·35**	**6·35**
finance512	2504	2639	**11·91**	**11·94**
ford1	252	303	**2·35**	2·61
ford2	956	961	**41·05**	**41·66**
nasasrb	540	864	**19·01**	**18·63**
onera_dual	2712	5074	**87·75**	103·41
pds10	3290	3419	**9·36**	12·53
shuttle_eddy	167	225	0·62	**0·59**
skirt	1776	1879	36·60	**34·16**
tandem_dual	2139	2331	**66·21**	72·98
tandem_vtx	1472	1774	**5·72**	**5·75**

Table V. Percentage increases in profiles for different weights

Weights	*barth*5	*copter*2	*onera_dual*	*finance*512	*ford*1	*skirt*
(1, 64)	100·3	53·5	55·0	32·1	10·9	23·0
(1, 32)	100·3	53·5	55·0	32·0	10·9	23·0
(1, 16)	100·3	53·7	55·0	32·1	10·9	22·6
(1, 8)	100·3	53·4	55·0	32·0	10·7	13·4
(1, 4)	100·4	51·3	55·0	30·6	9·8	2·7
(1, 2)	99·1	41·7	52·2	16·1	7·7	0·0
(1, 1)	88·5	26·4	41·7	7·4	4·2	3·3
(2, 1)	73·5	13·7	27·8	1·2	0·0	16·1
(4, 1)	47·8	7·5	12·2	0·0	0·9	46·4
(8, 1)	14·8	5·8	1·6	44·9	8·5	80·6
(16, 1)	1·5	0·0	0·0	318·0	10·3	130·7
(32, 1)	0·0	0·9	0·3	796·9	24·6	131·6
(64, 1)	0·0	0·9	0·3	954·4	25·8	131·7

Some examples illustrating our findings are shown in Table V, where percentage increases from the best value are shown. In both test sets, there are cases for which the profile rises rapidly for large values of W_1/W_2. Kumfert and Pothen call these problems *class two* and the rest *class one*.

The first three examples in Table V are class-one problems and the rest are class-two problems. The examples *barth5* and *finance512* were used by Kumfert and Pothen to exemplify classes one and two, respectively. We see from the table that, for class-one problems, it may be important

to choose a large value of W_1/W_2. For class-two problems, (1, 1) or Sloan's choice of (2, 1) both seem reasonable. Our results for the whole test set show that using the weights (2, 1) rarely gives profiles for class-two problems that are more than 5 per cent bigger than the best of those we computed. From the Kumfert and Pothen test set, the class-two problem for which the Sloan choice gave the worst result was *skirt*. Results for this problem are given in the final column of Table V. For class-one problems, it seems to be rarely advantageous to go beyond a ratio of 16, and can be slightly disadvantageous. Kumfert and Pothen have no suggestion for predicting to which class a problem belongs. It seems to us that, if the class of problem is not known, it is necessary to try more than one pair of weights. To allow for this, in our code MC60 the weights are input parameters which must be set by the user. The default option in the driver MC61 is to compute orderings for pairs (2, 1) and (16, 1) and to choose the one with the smallest profile. MC61 also allows the user to specify other choices for the weights.

Both Sloan and Kumfert and Pothen use an integer priority function, but this seems to us to be an unnecessary restriction. We use real values, which have the same storage requirement in the usual case of 4-byte integers and 4-byte reals. We found that there was an increase in execution time, but it was very slight. Using reals means that no tests are needed to ensure that integer overflow does not occur.

Sloan's algorithm will generally avoid very large semibandwidths simply because of not departing far from the underlying rooted level-set structure ordering, but this may give an ordering that is not satisfactory for an out-of-core variable-band solver. One possibility is to increase the weight W_1, but the direct use of the Reverse Cuthill–McKee order is likely to be better from this point of view.

4.6. Other adjustments of the priority function

Kumfert and Pothen [5] point out that the current degree c_i varies between 0 and $\Delta + 1$, where Δ is the maximum degree of a node, while $d(i, e)$ varies between 0 and h, the level-set depth. They therefore suggests replacing (1) by the priority function

$$P_i = -W_1 \operatorname{int}(h/\Delta)c_i + W_2 d(i, e) \tag{2}$$

Our feeling is that it is inappropriate to take the depth into account. What matters is the local nature of the graph and, for the second term in P_i, it is to which level sets the candidate nodes belong. This varies by one from each level set to the next so is already properly normalized.

Using (2), we have examined the profile sizes for the 13 pairs of weights used in the previous section on all the Everstine and Kumfert and Pothen test matrices without seeing any evidence that normalization is needed. Therefore, we do not use (2) in our implementation of Sloan's algorithm.

Strictly speaking, equations (1) and (2) do not define the priority function fully since we give maximum, priority to a node with $c_i = 0$. Thus the priority function is really non-linear in c_i. Nick Gould suggests (Private Communication) that further non-linearity might be helpful, but we have not investigated this.

4.7. Other start and end nodes

Recall from Section 3.3 that Paulino *et al.* suggested using the spectral method to find a pseudodiameter and then using the better of the RCM orderings based on the two ends of this

Table VI. Profiles (in millions) with different algorithms

Problem	Sloan MC40	Sloan MGPS	Sloan spectral	Hybrid perm.	Hybrid vector	Spectral
barth	0·49	0·47	0·48	**0·40**	**0·40**	0·46
barth4	0·45	0·33	0·37	**0·29**	**0·29**	0·33
barth5	2·43	1·44	1·48	**1·29**	**1·29**	1·42
bcsstk30	15·72	16·15	14·86	**7·88**	8·20	9·14
commanche_dual	0·44	**0·33**	**0·34**	0·35	0·35	0·35
copter1	7·09	**6·05**	**6·05**	6·11	6·11	7·61
copter2	43·24	37·96	35·28	**32·78**	**32·78**	42·00
finance256	6·57	**6·35**	6·51	**6·44**	6·70	9·17
finance512	12·22	11·91	14·25	11·72	**11·41**	19·13
ford1	2·34	2·35	2·74	1·95	**1·88**	2·17
ford2	40·63	41·05	41·78	**35·97**	**35·64**	40·30
nasasrb	**18·35**	19·01	19·38	19·30	19·21	25·10
onera_dual	113·67	87·75	81·88	**46·67**	**46·66**	53·39
pds10	13·68	9·36	9·87	**8·81**	**8·89**	16·06
shuttle_eddy	**0·59**	0·62	0·62	**0·59**	**0·59**	0·76
skirt	34·12	36·60	33·38	**27·87**	29·26	30·51
tandem_dual	87·79	66·21	79·85	**42·22**	**42·22**	48·38
tandem_vtx	6·29	5·72	5·46	**5·22**	**5·22**	6·19

pseudodiameter. We can also use the spectral pseudodiameter to give start and target end nodes (s, e) for the numbering phase of Sloan's method. We again choose the start node to be the end of the pseudodiameter which gives the narrowest level-set structure. This is not a big overhead as it just requires one more level-set structure to be constructed. We found the overall quality of the results to be very similar to those obtained using the MGPS pseudodiameter—spectral start and end nodes were better on some problems and worse on others. This is illustrated by the results presented in column 'Sloan Spectral' of Table VI.

5. THE HYBRID METHOD

Kumfert and Pothen [5] observed that spectral orderings do well in a global sense but often do poorly locally. They therefore proposed using the spectral method to provide a global ordering to guide Sloan's method. Their results showed that this can yield a much better final ordering than using either the spectral method alone or Sloan's method with the rooted level-set structure ordering. Kumfert and Pothen propose the priority function

$$P_i = -W_1 \operatorname{int}(n/\Delta)c_i + W_2 d(i,e) - W_3 p_i \tag{3}$$

where p_i is the position of node i in the spectral ordering and call this the *hybrid* method. The normalization has been changed to balance the maximum values of the factors for W_1 and the new W_3. They use the spectral pseudodiameter to find the end node e and leave the distance $d(i,e)$ unnormalized, which gives the second term in (3) only a small influence. Although in their paper they report that choosing W_2 to be equal to one generally does significantly better than setting W_2 to zero, they later say (Private Communication) that they regret including this term.

To make the normalization similar to that of the priority function (1), in place of (3) we have chosen to use the priority function

$$P_i = -W_1 c_i - W_2(h/n)p_i \tag{4}$$

where h is the level-set depth. This makes the factor for W_2 vary up to h, as in (1). Our results are summarized in the column 'Hybrid perm.' of Table VI. For some problems, including *tandem_dual* and *onera_dual*, we found a significant improvement by using the whole spectral order as a guide for Sloan's method. It appears that the spectral ordering of the interior nodes is important. We see no possible justification for using both the level-set order and the spectral order to guide the Sloan algorithm and our results are comparable with those reported by Kumfert and Pothen. We again tried a range of values for W_1 and W_2. For class-two problems, we found that a value of W_1/W_2 that was smaller than that used by Sloan was advantageous. Based on our numerical experiments, for the hybrid method for this class, we recommend using the weights (1, 2) rather than (2, 1). In Table VI, for the hybrid method we take the better of the results for the weights (1, 2) and (16, 1). In general, the best weight for p_i must depend on the quality of the permutation and the higher weight that we have found useful with the spectral order indicates that it is of good quality.

For the hybrid method, we again experimented with interchanging the ends of the pseudodiameter. We constructed the level-set structures rooted at the two ends and selected as the starting node the one with the narrowest level structure. We found that for some problems this gave a reduction in the profile but for other problems, it gave an increase. We do not think the differences in the profile are large enough to justify the expense of running the Sloan algorithm from both nodes so in our code we use only one.

We have also tried using the Fiedler vector from the spectral method directly, again adjusting the normalization so that the factor for W_2 in (4) varies up to the depth h. We found (see column 'Hybrid vector' of Table VI) that overall this did not significantly improve the quality of the results. We also show in Table VI (column 'Spectral') the profiles for the spectral ordering. A comparison of columns 5 and 7 demonstrate that it is worthwhile to use Sloan's method to refine the spectral ordering.

In Table VI, we highlight in bold the smallest profile for each problem and any within 2 per cent of the smallest. We also show MC40 profiles for these problems. Note that tie-breaking can affect all these results so that too much notice should not be taken of small differences. For example, MC40 gives slightly better profiles than Sloan MGPS on six problems but generally its results are less good because it uses only the weights (2, 1).

The hybrid method was intended for very large problems, but we felt that it would be of interest to see how it performs on some of the Everstine problems, since these have been widely used as a test set and very good profiles have been obtained for them by Armstrong [19] using simulated annealing (which would not be suitable for everyday software). In Table VII, we compare Armstrong's profiles with those obtained with the hybrid method and with the Sloan MGPS method. On this size of problem, there seems to be a little advantage in using the hybrid algorithm; the profiles are within 2 per cent of each other in five cases, and each is significantly better than the other in two cases. In one case, however, the hybrid profile is less than Armstrong's.

We remark that although we have only used the spectral ordering in the hybrid algorithm, any input ordering can be used. Our codes are written to allow this.

Table VII. Profiles (in thousands) for the three algorithms on the nine largest Everstine problems

n	Sloan MGPS	Hybrid perm.	Armstrong
758	7·3	7·5	7·1
869	13·9	15·7	13·2
878	19·4	19·2	17·8
918	17·0	17·3	15·9
992	33·5	33·4	32·5
1005	34·7	30·8	32·5
1007	22·7	20·4	19·9
1242	36·5	39·8	33·1
2680	89·7	91·4	84·9

6. SOFTWARE DESIGN

In this section, we discuss the design of new codes for reordering sparse symmetric matrices. Our new subroutines are named according to the naming convention of the Harwell Subroutine Library [20]. The codes themselves are available; please contact one of the authors for details of price and conditions of use.

Our previous code MC40 provided the user with a single subroutine. It accepted the strictly lower triangular part of the matrix and returned the permutation and the values of the profiles for the original and permuted orderings. While the design of our new software includes a simple driver, MC61, we have decided that it is very worthwhile to give the user the greatest possible flexibility so in the MC60 package we provide user entries to the component parts of the reordering algorithm. These are described in the following subsections.

6.1. MC60A

MC60A accepts the pattern of the lower-triangular part of the matrix **A** and constructs the pattern of the whole matrix. Each is held by columns, with pointers to the column starts. For economy of storage, the work is done in place. A first pass looks for any out-of-range or repeated indices and removes them, or terminates if this has been requested. A second pass counts the number of entries that need to be added to each row to include the upper triangle. A third pass works through the rows in reverse order, moving them back to allow space for the additional entries. A final pass inserts the additional entries. There are extensive checks on the data. If the user already has the pattern of the whole matrix and does not wish checks to be made on the data, MC60A is not needed.

6.2. MC60B

MC60B constructs supervariables, given the pattern of the whole matrix. This is done in $O(n + \tau)$ time, where n is the order of the matrix and τ is the number of entries, by working

progressively so that after j steps we have the supervariable structure for the submatrix of the first j columns. We start with all variables in one supervariable (for the submatrix with no columns), then split it into two according to which rows do or do not have an entry in column 1, then split these according to the entries in column 2, etc. The splitting is done by moving the variables one at a time to the new supervariable. Further details are given by Duff and Reid [21].

Note that this strategy requires the user to provide the indices of the entries on the diagonal since these affect whether the structures of columns are identical. This contrasts with MC40, which assumes that the diagonal entries are all non-zero.

Unless all the supervariables consists of a single variable, we now compress the pattern. This is held in the form of the index list for column 1, followed by the index list for column 2, etc. with pointers to column starts. In order that this compression can be performed in place, we identify for each supervariable the member variable with least index as its key variable. This permits us to visit the supervariables in the order of their key variables, picking up the index list of that variable, using it to construct the supervariable index list, and placing the constructed list in its final position without fear of overwriting any information needed later. Note that the supervariable pattern and the map of variable to supervariable indices provides a complete representation of the original pattern. For efficiency, we also hold an array of numbers of variables in supervariables.

The use of MC60B is optional. If it is known that there are few supervariables, MC60B will not be needed. On the other hand, MC60B may be used in combination with another algorithm for choosing an ordering.

6.3. MC60C

MC60C controls the main part of the algorithm. It works with the supervariable graph (Section 3.4) and returns a supervariable permutation. It allows for the matrix being reducible (a permutation of a block diagonal matrix). In this case, each diagonal block of the permuted matrix will correspond to a component of the graph (set of nodes with no connections to other nodes). It orders any trivial components first by choosing any nodes that have degree zero. It then orders each non-trivial component in turn by calling other subroutines, which allows these other subroutines to work with a single component.

The user has to choose whether Sloan's algorithm or RCM is required for each component. For Sloan's algorithm, the user may specify a global priority vector whose components p_i are used in the priority function (4). This will normally come from a spectral ordering, but is not restricted to this. Apart from this case, a pseudodiameter must be found.

By default, MC60C calls MC60H to compute a pseudodiameter, as explained in Section 3.1. MC60H also distinguishes between the ends, as explained in Section 3.2. Alternatively, the user may specify the two end nodes. In either case, the level-set structure rooted on the end node is needed to provide distances to the end node in Sloan's method and the ordering itself for the RCM method. This is always returned by MC60H; if the end nodes are specified, the level-set structure is computed by calling MC60L directly from MC60C (MC60L is also called from MC60H).

It is this need for the level-set structure that leads to the choice of end node possibly adding an overhead (see Section 3.2). If we do not mind which end is used and if the most recent level-set structure was constructed in full (the construction is terminated early if its width is found to be greater than the narrowest encountered so far, see Section 3.1), a final call of MC60L is not needed.

If Sloan's method is required, this is performed by MC60J, using weights W_1 and W_2 supplied by the user and switching from a simple search to using a binary heap if the front gets large (see Section 4.2). If RCM is required, we have only to reverse the order that we already have.

6.4. *MC60D, MC60E, MC60F, and MC60G*

MC60D, MC60E, MC60F, and MC60G are simple utilities that convert the supervariable ordering to an ordering for variables or rows, or provide statistics.

MC60D constructs the permutation for the variables that corresponds to a given permutation for the supervariables.

MC60E uses a given permutation for the supervariables to construct the corresponding ordering for the rows, as required by a row-by-row frontal solver such as MA42 (equation entry).

MC60F uses a given permutation for the supervariables to compute the profile, the maximum wavefront, the semibandwidth, and the root-mean-square wavefront for the permuted matrix.

MC60G uses a given row order to compute the maximum row and column front sizes and the root-mean-square row and column front sizes for a row-by-row frontal method.

6.5. *The driver MC61*

For our driver MC61, there are just two entries. The subroutine MC61I must be called to provide default values for the parameters that control the execution of the package. If the user wishes to use values other than the defaults, the corresponding parameters should be reset after the call to MC61I. MC61A accepts the pattern of the lower-triangular part of **A**, performs full checks on the data, and either

(1) chooses a permutation of the variables that aims to reduce the profile and wavefront of the matrix,
(2) chooses a permutation of the variables that aims to reduce the bandwidth of the matrix, or
(3) constructs an ordering for the rows that is efficient when used with a row-by-row frontal solver.

Although MC61 has a user interface which is similar to that of MC40, it provides a much wider range of options. The user may choose whether or not to use supervariables. The user may also specify the weights for the Sloan priority function (1) and can optionally supply the vector $\{p_i\}$ and weights for the hybrid priority function (3).

7. CONCLUDING DISCUSSION

In this paper, we have discussed the design and development of a software package, MC60, for computing a symmetric permutation to reduce the profile and wavefront of a large sparse matrix with a symmetric sparsity pattern. The driver MC61 provides the user with a straightforward interface to MC60. In the next release (HSL 2000) of the Harwell Subroutine Library, MC61 will supersede MC40.

As we discussed in Section 4.5, if Sloan's algorithm (combined with the MGPS algorithm for finding a pseudodiameter) is selected, we recommend computing profiles for the pairs of weights (2, 1) and (16, 1) and taking the best. This is the default option in MC61. Although the

Table VIII. Timings for MC40 and MC61. Nsup denotes the number of supervariables Var. indicates that variables are used, Svar. indicates that supervariables are used

				MC61			
				One pair weights		Two pairs weights	
Problem	Order	Nsup	MC40	Var.	Svar.	Var.	Svar.
barth	6691	6691	0·17	0·18	0·21	0·28	0·31
barth4	6019	6019	0·12	0·15	0·14	0·21	0·24
barth5	15 606	15 606	0·56	0·45	0·52	0·71	0·78
bcsstk30	28 924	**9289**	6·29	3·82	1·91	6·40	2·28
commanche_dual	7920	7920	0·13	0·13	0·16	0·22	0·24
copter1	17 222	17 222	1·44	0·57	0·70	0·99	1·09
copter2	55 476	55 476	11·55	2·39	2·80	4·12	4·52
finance256	37 376	37 376	1·93	1·10	1·29	1·85	2·04
finance512	74 752	74 752	3·41	2·12	2·50	3·57	3·95
ford1	18 728	18 210	0·60	0·38	0·44	0·65	0·69
ford2	100 196	97 906	8·68	2·63	2·92	4·39	4·60
nasasrb	54 870	**24 954**	4·59	5·05	2·99	8·51	3·85
onera_dual	85 567	85 567	21·83	2·50	2·84	4·16	4·51
pds10	16 558	16 558	5·40	0·56	0·65	0·92	1·02
shuttle_eddy	10 429	10 363	0·22	0·30	0·35	0·47	0·53
skirt	45 361	**14 956**	8·97	4·83	2·58	8·20	3·17
tandem_dual	94 069	94 069	16·96	2·57	2·90	4·12	4·51
tandem_vtx	18 454	18 454	1·33	0·81	0·94	1·27	1·40

pseudodiameter does not need to be recomputed, the use of two pairs of weights does represent an overhead. To illustrate this and to compare the efficiency of the old and new ordering codes, in Table VIII we report timings for MC40 and MC61 for the Kumfert and Pothen test examples. For MC61 we show times for a single pair of weights and for two pairs of weights, using variables and using supervariables. The results in column 8 are those for the default MC61 parameters.

Kumfert and Pothen report that the hybrid method is more than six times more expensive than the Sloan method. In this study we do not include timings for the hybrid method because the Chaco package that we use to find the Fiedler vector is written in C and we do not currently have a Fortran code within the Harwell Subroutine Library for computing the Fiedler vector.

As anticipated, for the larger problems, using the binary heap gives significant savings so that, for these problems, MC61 with two pairs of weights is still generally faster than MC40. For the smaller problems, MC61 can be slower than MC40, but the quality of the MC61 ordering is usually superior. Only three of the test problems, *skirt*, *nasarb*, and *bcsstk30*, have significantly fewer supervariables than variables (highlighted in bold). For these problems, we see there is a substantial saving in the execution times when supervariables are used. For the other problems, searching for supervariables increases the reordering time but the increase is generally limited to about 15 per cent for a single pair of weights and about 10 per cent for two pairs. We therefore recommend that, unless the user knows his or her problem does not compress well, supervariables with two pairs of weights should be used, and this is the default in MC61.

ACKNOWLEDGEMENTS

We are grateful to Gary Kumfert and Alex Pothen of Old Dominion University and to Scott Sloan of the University of Newcastle, New South Wales for helpful discussions. We would like to thank Gary Kumfert for providing us with the test examples used in this paper. Finally, we would like to thank Scott Sloan, Alex Pothen, and our colleagues Iain Duff and Nick Gould for their helpful comments on drafts of this paper.

REFERENCES

1. Paulino GH, Menezes IFM, Gattass M, Mukherjee S. A new algorithm for finding a pseudoperipheral vertex, or the endpoints of a pseudodiameter in a graph. *Communications in Numerical Methods in Engineering* 1994; **10**:913–926.
2. Sloan SW. An algorithm for profile and wavefront reduction of sparse materials. *International Journal for Numerical Methods in Engineering* 1986; **23**:239–251.
3. Sloan SW. A Fortran program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering* 1989; **28**:2651–2679.
4. Duff IS, Reid JK, Scott JA. The use of profile reduction algorithms with a frontal code. *International Journal for Numerical Methods in Engineering* 1989; **28**:2555–2568.
5. Kumfert G, Pothen A. Two improved algorithms for envelope and wavefront reduction. *BIT* 1997; **18**:559–590.
6. Barnard ST, Pothen A, Simon H. A spectral algorithm for envelope reduction of sparse matrices. *Numerical Linear Algebra with Applications* 1995; **2**:317–334.
7. Gibbs, NE, Poole WG, Jr, Stockmeyer PK. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal for Numerical Analysis* 1976; **13**:236–250.
8. Everstine GC. A comparison of three resequencing algorithms for the reduction of matrix profile and wavefront. *International Journal for Numerical Methods in Engineering* 1979; **14**:837–853.
9. Duff IS, Erisman AM, Reid JK. *Direct Methods for Sparse Matrices*. Oxford University Press: London, 1986.
10. Duff IS, Scott JA. The design of a new frontal code for solving sparse, unsymmetric systems. *ACM Transactions on Mathematical Software* 1996; **22**:30–45.
11. Gibbs NE. A hybrid profile reduction algorithm. *ACM Transactions on Mathematical Software* 1976; **2**:378–387.
12. Sloan SW, Randolph MF. Automatic element reordering for finite element analysis with frontal solution schemes. *International Journal for Numerical Methods in Engineering* 1983; **19**:1153–1181.
13. George A, Liu JWH. An implementation of a pseudoperipheral node finder. *ACM Transactions on Mathematical Software* 1979; **5**:284–295.
14. Lewis JG. Implementation of the Gibbs–Poole–Stockmeyer and Gibbs–King algorithms. *ACM Transactions on Mathematical Software* 1982; **8**:180–189 and 190–194.
15. Cuthill E, McKee J. Reducing the bandwidth of sparse symmetric matrices. *Proceedings 24th National Conference of the Association for Computing Machinery*, Brandon Press: NJ, 1969; 157–172.
16. George A. Computer implementation of the finite-element method. *Report STAN CS-71-208, Ph.D. Thesis*, Department of Computer Science, Stanford University, Stanford, CA, 1971.
17. Paulino GH, Menezes IFM, Gattass M, Mukherjee S. Node and element resequencing using the Laplacian of a finite element graph: parts I and II. *International Journal for Numerical Methods in Engineering* 1994; **37**:1511–1555.
18. Hendrickson B, Leland R. The Chaco user's guide: Version 2.0, *Technical Report* SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1995.
19. Armstrong BA. Near-minimal matrix profiles and wavefronts for testing nodal resequencing algorithms. *International Journal for Numerical Methods in Engineering* 1985; **21**:1785–1790.
20. HSL. *Harwell Subroutine Library Catalogue* (*Release 12*). AEA Technology: Harwell, 1995.
21. Duff IS, Reid JK. Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software* 1996; **22**:227–257.

