RAL-TR-98-056

A new row ordering strategy for frontal solvers.*

by

Jennifer A. Scott

Abstract

The frontal method is a variant of Gaussian elimination that has been widely used since the mid 1970s. In the innermost loop of the computation the method exploits dense linear algebra kernels, which are straightforward to vectorize and parallelize. This makes the method attractive for modern computer architectures. However, unless the matrix can be ordered so that the front is never very large, frontal methods can require many more floating-point operations for factorization than other approaches. We use the idea of a row graph of an unsymmetric matrix combined with a variant of Sloan's profile reduction algorithm to reorder the rows. We also look at using the spectral method applied to the row graph. Numerical experiments are performed on a range of practical problems. Our new row ordering algorithm is shown to produce orderings that are a significant improvement on those obtained with existing algorithms. Numerical results also compare the performance of the frontal solver `MA42` on the reordered matrix with other direct solvers for large sparse unsymmetric linear systems.

Keywords: ordering rows, frontal method, row graphs.

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 sRAL98056.ps.gz.

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

August 6, 1998.

Contents

1 Introduction

The frontal method is a technique for the direct solution of linear systems

$$Ax = b, \tag{1.1}$$

where the $n \times n$ matrix A is large and sparse. Although the method was originally developed for the solution of finite-element problems in which A is a sum of elemental matrices (see Irons, 1970, Hood, 1976), it can be used to solve any general linear system of equations (Duff, 1981, 1984). In this paper, we are concerned with using the frontal method for unsymmetric non-element problems; in a separate paper (Scott, 1998) we discuss ordering strategies for element problems.

The frontal method is a variant of Gaussian elimination that involves computing the decomposition of a permutation of A

$$PAQ = LU,$$

where L is unit lower triangular and U is upper triangular. A key feature of the method is that, at each stage of the computation, only a subset of the rows and columns of A needs to be held in main memory, in a matrix termed the *frontal matrix*. The rows of A are assembled into the frontal matrix in turn. Column l is defined as being *fully summed* once the last row with an entry in column l has been assembled. A column is *partially summed* if it has an entry in at least one of the rows assembled so far but is not yet fully summed. Once a column is fully summed, partial pivoting is performed to choose a pivot from that column.

At each stage, the frontal matrix F is a rectangular matrix. The number of rows in the frontal matrix is the *row frontsize* and the number of columns the *column frontsize*. Assuming there are k fully summed columns (with $k \geq 1$) and assuming the rows of F have been permuted so that the pivots lie in positions $(1,1)$, $(2,2)$, ..., (k,k), the frontal matrix can be written in the form

$$F = \begin{pmatrix} F_1 & F_2 \end{pmatrix}, \quad F_1 = \begin{pmatrix} F_{11} \\ F_{21} \end{pmatrix}, \quad F_2 = \begin{pmatrix} F_{12} \\ F_{22} \end{pmatrix}, \tag{1.2}$$

where F_{11} is of order $k \times k$. The columns of F_1 are fully summed while those of F_2 are partially summed. If F_{12} is of order $k \times m$ and F_{21} is of order $l \times k$, the row and column frontsizes are $k + l$ and $k + m$, respectively. F_{11} is factorized as $L_{11}U_{11}$. Then F_{12} and F_{21} are updated as

$$\hat{F}_{21} = F_{21}U_{11}^{-1} \quad \text{and} \quad \hat{F}_{12} = L_{11}^{-1}F_{12} \tag{1.3}$$

and then the Schur complement

$$F_{22} - \hat{F}_{21}\hat{F}_{12} \tag{1.4}$$

is formed. Finally, the factors L_{11} and U_{11}, as well as $\hat{F}_{12}$ and $\hat{F}_{21}$, are stored as parts of L and U, before further rows from the original matrix are assembled with the Schur complement to form another frontal matrix.

The power of frontal schemes comes from the fact that they are able to solve quite large problems with modest amounts of main memory and the fact that they are able to perform the numerical factorization using dense linear algebra kernels, in particular the Level 3 Basic Linear Algebra Subprograms (BLAS) (Dongarra, DuCroz, Duff and Hammarling, 1990) may be used. For example, the BLAS routine `GEMM` with interior dimension k can be used to form the Schur complement (1.4).

Since a variable can only be eliminated after its column is fully summed, the order in which the rows are assembled will determine both how long a variable remains in the front and the order in which the variables are eliminated. For efficiency, in terms of both storage and arithmetic operations, the rows need to be assembled in an order that keeps both the row and column frontsizes as small as possible. If $frow_i$ and $fcol_i$ denote the row and column frontsizes before the ith elimination, we are interested in

- the maximum row and column frontsizes

$$frow_{max} = \max_{1 \le i \le n} frow_i \quad \text{and} \quad fcol_{max} = \max_{1 \le i \le n} fcol_i \tag{1.5}$$

  since these determine the amount of main memory needed,

- the root-mean-square row and column frontsizes

$$frow_{rms} = \frac{1}{n}\sqrt{\sum_{i=1}^{n} {frow_i}^2} \quad \text{and} \quad fcol_{rms} = \frac{1}{n}\sqrt{\sum_{i=1}^{n} {fcol_i}^2} \tag{1.6}$$

  since these provide a measure of the average row and column frontsizes,

- the average size of the frontal matrix

$$\frac{1}{n}\sum_{i=1}^{n}(frow_i * fcol_i). \tag{1.7}$$

  A prediction of the number of floating-point operations that must be performed can be obtained from (1.7) (assuming zeros within the frontal matrix are not exploited).

Because reordering aims to reduce the length of time each variable is in the front, we introduce the concept of the lifetime of a variable. For a given ordering, the lifetime $Life_i$ of variable i is defined to be $last_i - first_i$, where $first_i$ and $last_i$ are the assembly step when variable i enters and leaves the front, respectively. That is,

$$Life_i = \{\max_{1 \le k,l \le n} |l - k| : a_{kj} \neq 0 \text{ and } a_{lj} \neq 0\}. \tag{1.8}$$

A useful measure is the sum of the lifetimes: a small value for the sum of the lifetimes indicates we have a good row ordering.

We observe that, if A has a full row, the maximum row and column frontsizes will be n, irrespective of the order in which the rows of A are assembled.

Similarly, if A has one or more rows that are almost full, the maximum column frontsize will be large. Clearly, the frontal method is not a good choice for such systems.

Throughout this study, we shall be concerned only with running the frontal method on a single processor. Different ordering strategies are required when implementing a frontal algorithm in parallel. This is discussed, for example, by Camarda (1997) and, for element problems, by Scott (1996), and remains a subject for further investigation.

The outline of this paper is as follows. In Section 2, we recall some basic concepts from graph theory and, in particular, introduce the idea of the row graph of an unsymmetric matrix. We briefly review existing row ordering strategies in Section 3 and, in Section 4, explain the Cuthill-McKee and Sloan algorithms for reordering the nodes of an undirected graph. Our new reordering algorithms are introduced in Section 5. We look at applying Cuthill-McKee and variants of Sloan's algorithm to the row graph of A, and also introduce a hybrid algorithm that uses the spectral algorithm, again applied to the row graph. Extensive numerical results illustrating the effectiveness of our new algorithms are presented in Section 6. In Section 7, we use the new algorithms with a frontal solver and, in Section 8, we compare the performance of our frontal solver used with our new reordering algorithms with that of other sparse direct solvers. Finally, some concluding remarks are made in Section 9.

2 Graphs and matrices

Before looking at row ordering algorithms, it is convenient to recall some basic concepts from graph theory.

A *graph* G is defined to be a pair (V, E), where V is a finite set of *nodes* (or *vertices*) $v_1, v_2, ..., v_n$, and E is a finite set of *edges*, where an edge is a pair (v_i, v_j) of distinct nodes of V. If no distinction is made between (v_i, v_j) and (v_j, v_i) the graph is *undirected*, otherwise it is a *directed graph* or *digraph*. A *labelling* (or *ordering*) of a graph $G = (V, E)$ with n nodes is a bijection of $\{1, 2, ..., n\}$ onto V. The integer i $(1 \leq i \leq n)$ assigned to a node in V by a labelling is called the *label* (or *number*) of that node. Two nodes v_i and v_j in V are said to be *adjacent* (or *neighbours*) if $(v_i, v_j) \in E$. The *degree* of a node $v_i \in V$ is defined to be the number of nodes in V which are adjacent to v_i, and the *adjacency list* for v_i is the list of these adjacent nodes. A *path of length* k in G is an ordered set of distinct nodes $(v_{i_1}, v_{i_2}, ..., v_{i_{k+1}})$ where $(v_{i_j}, v_{i_{j+1}}) \in E$ for $1 \leq j \leq k$. Two nodes are *connected* if there is a path joining them. An undirected graph G is *connected* if each pair of distinct nodes is connected. Otherwise, G is disconnected and consists of two or more *components*.

We now establish the relationship between graphs and matrices. A labelled graph $G(A)$ with n nodes can be associated with any square matrix $A = \{a_{ij}\}$ of order n. Two nodes i and j $(i \neq j)$ are adjacent in the graph if and only if a_{ij} is nonzero. If A has a symmetric sparsity pattern, $G(A)$ is undirected, otherwise $G(A)$ is a digraph. The graph of a symmetric matrix is unchanged if a symmetric permutation is performed on the matrix; only the labelling of

its nodes changes. Many reordering algorithms for sparse symmetric matrices exploit the close relationship between the matrix and its undirected graph (for example, the algorithms of Cuthill and McKee, 1969, and Sloan, 1986, which we discuss briefly in Section 4).

The *bipartite graph* of A consists of two distinct sets of n nodes R and C, each set being labelled $1, 2, ..., n$, together with E edges joining nodes in R to those in C. There is an edge between $i \in R$ and $j \in C$ if and only if a_{ij} is nonzero. Here, $|E|$ is the total number of entries in A.

In this study, we are concerned with row permutations of unsymmetric matrices. We could use a digraph or a bipartite graph. However, reordering techniques for undirected graphs have been the subject of much research and we would like to exploit some of these techniques. This motivates us to consider using row graphs. Row graphs were introduced by Mayoh (1965). The *row graph* G_R of A is defined to be the undirected graph of the symmetric matrix $B = A * A^T$, where $*$ denotes matrix multiplication without taking cancellations into account (so that, if an entry in B is zero as a result of numerical cancellation, it is considered as a nonzero entry and the corresponding edge is included in the row graph). The nodes of G_R are the rows of A and two rows i and j $(i \neq j)$ are adjacent if and only if there is at least one column k of A for which a_{ik} and a_{jk} are both nonzero. Row permutations of A correspond to relabelling the nodes of the row graph. In Section 5, we consider how to reorder the nodes of G_R to produce row orderings that are appropriate for use with a frontal solver.

3 Existing row ordering strategies

In the past, a number of algorithms for automatically ordering rows for frontal solvers have been proposed. If the matrix A has a symmetric sparsity pattern, an appropriate ordering can be obtained using a profile reduction algorithm such as that of Sloan (1986) and Reid and Scott (1998) (see also Section 4). For matrices with an almost symmetric pattern, good orderings can generally be obtained by applying a profile reduction algorithm to the sparsity pattern of $A + A^T$. However, for matrices that have a highly unsymmetric structure other techniques are required.

For frontal methods, an upper triangular form may appear attractive because as each row enters the front, a variable is immediately available for elimination. One possible approach to reordering, therefore, is to use an algorithm such as the partitioned preassigned pivot procedure (P4) of Hellerman and Rarick (1972) for reordering a matrix to almost lower triangular form and then to reverse the order. This was proposed by Stadtherr and Vegeais (1985). In his thesis, Camarda (1997) reports that reverse P4 gives inconsistent results in that, for some examples, it can produce good orderings but for other problems, it can give orderings that are significantly worse than the original ordering. This is possibly because the method places the rows with the largest number of entries early in the ordering and this can lead to a large column frontsize for many elimination steps. It is clear that, rather than a triangular form, a (variable) band form is needed for the frontal method to be efficient.

The restricted minimum column degree (RMCD) ordering algorithm for reducing the size of the frontal matrix was recently discussed by Camarda (1997). This algorithm uses the concept of a net. A *net* is defined to be a column l and all the rows i such that a_{il} is nonzero. This concept is useful because, when a net has been assembled, column l is fully summed and an elimination can be performed. At each stage of reordering, the degree of a column l is the number of nonzero entries a_{il} in the rows of A that have not yet been reordered. The RMCD algorithm stores the degree of each column and, at each stage, chooses the column of minimum degree and assembles all the rows in the net corresponding to the chosen column into the frontal matrix. The column degrees are then updated before the next column is selected. Rapid determination of the column with minimum degree is achieved through the use of linked lists. When the degree of a column is updated, the column is placed at the head of the linked list of columns of that degree. Thus partially summed columns are given priority. In his numerical experiments, Camarda found that the reordering time required by the RMCD algorithm was generally small compared with the time required by the subsequent numerical factorization of the matrix and the method gave modest improvements to the row ordering for a significant number of test examples.

The RMCD algorithm is a local heuristic ordering: at each stage it chooses the column of minimum degree without reference to effects on later stages. An alternative is to use an approach based on global heuristics, such as the recursive graph partitioning algorithm introduced by Coon (1990) and Coon and Stadtherr (1995) and modified by Camarda (1997). The goal of these algorithms is to find a partitioning of the bipartite graph of A such that the number of nets cut by the partition is minimised. The New Minimum Net Cut (NMNC) algorithm of Camarda is more expensive to implement than the simple RMCD algorithm but the results presented in the thesis of Camarda show that it can yield better orderings. This suggests that provided the number of factorizations following the initial reordering is sufficiently large, this method can be useful.

We have performed numerical experiments using both the RMCD and NMNC algorithms. Our findings, which are presented in Section 6 (Table 6.2), are consistent with those reported by Camarda (1997). They show that the performance of the RMCD algorithm can vary greatly between problems and, although more consistent, the NMNC algorithm generally is only able to achieve modest reductions in the size of the frontal matrix. New row ordering strategies are needed if we are to make the frontal method competitive with other methods for the direct solution of large sparse linear systems.

4 Bandwidth and profile reduction algorithms

In this section, we give a brief outline of the Reverse Cuthill-McKee algorithm and the Sloan algorithm for reordering the nodes of an undirected graph $G = (V, E), V = \{1, 2, ..., n\}$. Reverse Cuthill-McKee is primarily aimed at reducing the *bandwidth* B while Sloan's algorithm is designed to reduce the *profile* P,

where

$$B = \max_{(i,j)\in E} (i+1-j) \quad \text{and} \quad P = \sum_{i=1}^{n} \max_{j} \{i+1-j : (i,j) \in E\} \tag{4.1}$$

In the following, we assume that the graph G is connected. If not, it is straightforward to apply the algorithms to each component of G.

4.1 The Reverse Cuthill-McKee algorithm

The Cuthill-McKee algorithm divides the nodes into level sets. A *level structure rooted at a node* r is defined as the partitioning of V into levels $l_1(r), l_2(r), ..., l_{h(r)}$ such that

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

Cuthill-McKee orders within each level set $l_i(r)$ by ordering first nodes that are neighbours of the first node in $l_{i-1}(r)$, then those that are neighbours of the second node in $l_{i-1}(r)$, and so on. The Reverse Cuthill-McKee algorithm reverses the order found by Cuthill-McKee. This does not reduce the bandwidth further but can yield worthwhile reductions in the profile (see Liu and Sherman, 1976). The root r of the level structure is usually chosen to one of the ends of a pseudodiameter of G. The *distance* between nodes i and j in a G is denoted by $d(i,j)$, and is defined to be the number of edges on the shortest path connecting them. The *diameter* $D(G)$ of G is the maximum distance between any pair of nodes. That is,

$$D(G) = \max\{d(i,j) : i,j \in V\}. \tag{4.2}$$

A *pseudodiameter* $\delta(G)$ is defined by any pair of nodes i and j in G for which $d(i,j)$ is close to $D(G)$. A pseudodiameter may be found efficiently using a modified version of the Gibbs, Poole and Stockmeyer (1976) algorithm (see Sloan, 1989 and Reid and Scott, 1998).

4.2 The Sloan algorithm

Sloan's algorithm has two distinct phases:

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

In phase 1, a pseudodiameter of G is computed. One end s of the pseudodiameter is taken to be the *start* node and the other e is used as the *target end node*. In the second phase of the algorithm, the pseudodiameter is used to guide the reordering. 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. Sloan defines a node to be *active* if it is adjacent to a node that has already been renumbered but has not itself been renumbered. He aims to reduce the profile by reducing the number of nodes that are active at each stage and he does this by localized reordering. Sloan begins at the start node s and uses the priority function

$$P_i = -W_1 cdeg_i + W_2 d(i,e) \tag{4.3}$$

for node i, where W_1 and W_2 are positive integer weights, $cdeg_i$ (the *current degree*) is the number of nodes that will become active 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 *eligible nodes* are defined to be those that are active together their neighbours. A node has a high priority if it causes either no increase or only a small increase in the number of active nodes and is at a large distance from the target end node e. Thus, a balance is kept between the aim of keeping the number of active nodes small and including nodes that have been left behind (further away from e than other candidates).

5 New row ordering algorithms

We now return to the problem we are interested in, that is, the reordering of the rows of a general matrix A for use with a frontal solver.

5.1 The RCMRO algorithm

As mentioned earlier, if the matrix A has a variable band structure, the row and column frontsizes in the frontal method will be small. Recall that the row graph G_R of A is the undirected graph of $B = A * A^T$. If we apply the Reverse Cuthill-McKee algorithm to G_R the bandwidth of B will, in general, be reduced and thus A will also have a small bandwidth. Our first idea is, therefore, to generate a row ordering for the frontal method by applying Reverse Cuthill-McKee directly to G_R. We will call this algorithm the RCMRO algorithm.

5.2 The SRO algorithm

In place of the Reverse Cuthill-McKee algorithm, we can apply Sloan's algorithm to G_R. The nodes of G_R are the rows of A. Therefore, the first phase of the algorithm finds two rows of A that are at maximum (or almost maximum) distance apart. One of these rows, the start row, is chosen as the first row to be ordered (labelled), that is, the first row that will be assembled during the frontal method. At the start of the second phase of the algorithm, the current degree of each row is equal to its degree. In the row graph, the degree of row i is the number of rows j for which there is at least one column with entries in both rows i and j. Assuming A is not structurally singular, the degree of row i, deg_i, is at least $l-1$ where l is the number of entries in row i and, in general, $deg_i > l$. Once the first row has been ordered, its neighbours become active and the current degree of each neighbour is decreased by one.

The next candidate row for labelling will be an active row (at a distance of one from the start row) or a row that is itself adjacent to an active row (at a distance of two from the start row), and which causes either no increase or only a small increase in the number of active rows. There will be no increase in the number of active rows if the candidate row brings no new columns into the front and therefore causes no increase in the column frontsize.

The algorithm proceeds in this way, at each stage aiming to keep the number of active rows small whilst favouring rows that are at a small distance from the first row. We will refer to this scheme as the SRO algorithm.

5.2.1 Example

We now illustrate the SRO method using the matrix with the sparsity pattern given in Figure 5.1. The neighbours of each row are listed in Table 5.1. Initially $cdeg_i$ is the number of neighbours of row i. We will use weights $(W_1, W_2) = (2, 1)$. For the matrix in Figure 5.1, the lifetimes are given in Table 5.1 and the sum of the lifetimes is 22. We observe that the minimum possible value for the sum of the lifetimes is nz, the number of entries in A, which for example this is 15. The start and target end rows (s, e) are chosen to be $(4, 6)$ $(d(4, 6) = 3$

	1	2	3	4	5	6
1	x		x	x		
2		x		x	x	
3	x		x	x		x
4		x				
5				x	x	x
6						x

Figure 5.1: The original matrix.

Row	Neighbours	$Life_i$	$cdeg_i$	$d(i,6)$	P_i
1	2, 3, 5	3	3	1	-5
2	1, 3, 4, 5	3	4	2	-6
3	1, 2, 5, 6	3	4	1	-7
4	2	5	1	3	1
5	1, 2, 3, 6	4	4	1	-7
6	3, 5	4	2	0	-4

Table 5.1: Lists of neighbours and initial priorities for SRO method.

and $d(i, j) \leq 3$, $i, j = 1, 2, ..., 6)$, and the initial priorities are then computed (Table 5.1). Row 4 is ordered first. Its neighbour, row 2, then becomes active and its priority increases by W_1 to -4. At this stage, the list of eligible rows comprises row 2 and its unnumbered neighbours, rows 1, 3, 5. Of these, row 2 has the highest priority and is ordered next. The priorities of rows 1, 3, 5

	1	2	3	4	5	6	*Priority*
4		x					–
2		x		x	x		–
1	x		x	x			-3
3	x		x	x		x	-5
5				x	x	x	-5
6						x	-4

Figure 5.2: Partially ordered matrix.

are updated, resulting in the matrix of Figure 5.2. The remaining unnumbered rows are all now active and the one with the highest priority is row 1. On assembling row 1, the priority of its unnumbered neighbours, rows 3 and 5, increases by W_1. Both rows 3 and 5 now have priority -3 and the order in

	1	2	3	4	5	6	*Priority*
4		x					–
2		x		x	x		–
1	x		x	x			–
3	x		x	x		x	-3
5				x	x	x	-3
6						x	-4

Figure 5.3: Partially ordered matrix.

which they are assembled is arbitrary. Assuming row 5 is ordered first, we obtain the final reordered matrix given in Figure 5.4. The sum of the lifetimes

	1	2	3	4	5	6
4		x				
2		x		x	x	
1	x		x	x		
5				x	x	x
3	x		x	x		x
6						x

Figure 5.4: Final reordered matrix.

for the reordered matrix is 18.

5.3 The MSRO algorithm

The SRO algorithm attempts to reduce the number of rows that are active during the frontal method. Since a row is defined to be active if it is adjacent to a row that has already been assembled, a row is active if some of its columns are partially summed. Therefore, we indirectly reduce the number of columns

in the front by reducing the number of rows that are active. In an attempt to reduce directly both the row and column frontsizes, our second method again uses the first phase of Sloan's algorithm applied to the row graph to obtain start and target end rows and then uses a modified priority function

$$P_i = -W_1 rcgain_i + W_2 d(i,e). \tag{5.1}$$

Here $rcgain_i = rgain_i + cgain_i$, where $rgain_i$ and $cgain_i$ are the increases to the row and column frontsizes resulting from assembling row i next. Assembling a row into the frontal matrix causes the row frontsize to either increase by one, to remain the same, or to decrease. The row frontsize increases by one if no columns become fully summed, it remains the same if a single column becomes fully summed, and it decreases if more than one column becomes fully summed. The increase in the column frontsize is the difference between the number of column indices that appear in the front for the first time and the number that become fully summed. If this difference is negative, the column frontsize decreases. Hence, if s_i is the number of columns that become fully summed when row i is assembled,

$$rgain_i = 1 - s_i \tag{5.2}$$

and

$$cgain_i = newc_i - s_i, \tag{5.3}$$

where $newc_i$ is the number of new column indices in the front. It follows that

$$rcgain_i = 1 + newc_i - 2s_i \tag{5.4}$$

and a row has a high priority if it brings a small number of new columns into the front and results in a large number of columns becoming fully summed. We call this method the MSRO algorithm.

5.3.1 Example

We now illustrate the MSRO method, again using the matrix given in Figure 5.1 and weights $(W_1, W_2) = (2, 1)$. The start and target end rows (s, e) are chosen to be $(4, 6)$ and the initial priorities are given in Table 5.2. Note that initially $rcgain_i$ is just one more than the number of entries in row i.

As in the SRO method, row 4 is ordered first, followed by row 2. Row 2 brings columns 4 and 5 into the front. Since row 1 has an entry in column 4, its priority increases by W_1. The priority of rows 3 and 5 is also increased by W_1 and, because row 5 has entries in both columns 4 and 5, its priority increases by $2 * W_1$, resulting in the matrix of Figure 5.5.

Row 5 has the highest priority and is ordered next, bringing column 6 into the front. The priorities of rows 3 and 6, which have entries in column 6, are increased, giving the matrix in Figure 5.6.

We now order row 6. The priority of row 3 then increases so that it is ordered ahead of row 1. The final reordered matrix is given in Figure 5.7. The sum of the lifetimes for the reordered matrix is 16

Row	$rcgain_i$	$d(i,6)$	P_i
1	4	1	-7
2	4	2	-6
3	5	1	-9
4	2	3	-1
5	4	1	-7
6	2	0	-4

Table 5.2: Initial priorities for MSRO method.

	1	2	3	4	5	6	*Priority*
4		x					–
2		x		x	x		–
1	x		x	x			-5
3	x		x	x		x	-7
5				x	x	x	-3
6						x	-4

Figure 5.5: Partially ordered matrix.

	1	2	3	4	5	6	*Priority*
4		x					–
2		x		x	x		–
5				x	x	x	–
1	x		x	x			-5
3	x		x	x		x	-5
6						x	-2

Figure 5.6: Partially ordered matrix.

	1	2	3	4	5	6
4		x				
2		x		x	x	
5				x	x	x
6						x
3	x		x	x		x
1	x		x	x		

Figure 5.7: Final reordered matrix.

5.4 Spectral ordering algorithms

Spectral algorithms have been used in recent years for matrix profile and wavefront reduction. Barnard, Pothen and Simon (1995) describe a spectral algorithm that associates a Laplacian matrix L with a given matrix $S = \{s_{ij}\}$ with a symmetric sparsity pattern,

$$L = \{l_{ij}\} = \begin{cases} -1 & \text{if } i \neq j \text{ and } \quad s_{ij} \neq 0 \\ 0 & \text{if } i \neq j \text{ and } \quad s_{ij} = 0 \\ \sum_{i \neq j} |l_{ij}| & \text{if } i = j. \end{cases} \tag{5.5}$$

An eigenvector corresponding to the smallest positive eigenvalue of the Laplacian matrix is termed a *Fiedler vector*. The spectral permutation of the nodes of the undirected graph $G(S)$ is computed by sorting the components of a Fiedler vector into monotonically nonincreasing or nondecreasing order.

For matrices A with an unsymmetric sparsity pattern, we can apply the spectral method to the symmetric matrix $B = A * A^T$, whose undirected graph is the row graph G_R of A. The spectral permutation of the nodes of this graph yields a row ordering and we shall try using this with the frontal method. In our numerical experiments (see Section 6), we call this method the *spectral row reordering* algorithm.

5.5 A hybrid method

When ordering symmetrically structured matrices for a small profile, 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) 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, in place of the priority function (4.3), Reid and Scott (1998) use the priority function

$$P_i = -W_1 cdeg_i - W_2(h/n)p_i, \tag{5.6}$$

where $cdeg_i$ is again the current degree of node i and h is the number of level-sets in the level set structure rooted at the start node. The normalization of the second term results in the factor for W_2 varying up to h, as in (5.1). Without normalization, the second term would have a much larger influence.

In the present study, we are concerned with obtaining row orderings for unsymmetric matrices A for use with a frontal algorithm. We can extend the hybrid method to this class of problems by applying it to the row graph of A. We will consider two versions of the *hybrid row ordering* algorithm. Both will compute p_i by applying the spectral algorithm to $B = A * A^T$. The first will

then use the priority function (5.6) and the second will generalise (5.1) and use the priority function

$$P_i = -W_1 rcgain_i - W_2(h/n)p_i. \tag{5.7}$$

In our numerical experiments, we will call the resulting methods the hybrid SRO and hybrid MSRO algorithms.

We remark that, in the hybrid row ordering algorithms any input ordering can be used in place of the spectral ordering. However, in our numerical experiments we use only the spectral ordering.

6 Numerical results

In this section, we first describe the problems that we use for testing the row ordering algorithms discussed in this paper 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. The problems are all taken from either the Harwell-Boeing Collection (Duff, Grimes and Lewis, 1992) or the University of Florida Sparse Matrix Collection (Davis, 1997).

Identifier	Order	Number of entries	Description/discipline
bayer04	20545	159082	Chemical process simulation
bayer09	3083	21216	Chemical process simulation
bp1600	1600	4841	Basis matrix from LP problem
extr1	2837	11407	Dynamic simulation problem
gre1107	1107	5664	Simulation studies in computer systems
hydr1	5308	23752	Dynamic simulation problem
lhr07c	7337	156508	Light hydrocarbon recovery
lhr14c	14270	307858	Light hydrocarbon recovery
lns3937	3937	25407	Fluid flow modelling
meg1	2904	58142	Chemical process simulation
meg4	5860	46842	Chemical process simulation
nnc1374	1374	8606	Nuclear reactor core modelling
onetone2	36057	227628	Harmonic balance method, one-tone
orani678	2529	90158	Economic model
orsreg1	2205	14133	Oil reservoir simulation
pores3	532	3474	Oil reservoir simulation
rdist1	4134	94408	Reactive distillation problem
sherman3	5005	20033	Oil reservoir simulation
wang3	26064	177168	3D semiconductor device simulation
west2021	2021	7353	Chemical engineering

Table 6.1: The test problems.

The test codes 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.

6.2 A comparison of the methods

We first compare the performance of the RMCD, NMNC, RCMRO, SRO, and MSRO algorithms described in Sections 3 and 5. For the SRO and MSRO algorithms, we use the weights (W_1, W_2) equal to each of the 13 pairs $(1, 64)$, $(1, 32)$, $(1, 16)$, ..., $(1, 1)$, $(2, 1)$, ..., $(64, 1)$ and select the best result (in Section 6.5 we consider the sensitivity of the algorithms to the choice of the weights). In Table 6.2, the root-mean-square row and column frontsizes ($frow_{rms}$ and $fcol_{rms}$) are given. We highlight in bold the smallest values of $frow_{rms}$ and $fcol_{rms}$ for each problem (and any that are within 5 percent of the smallest). We also give the root-mean-square frontsizes for the original ordering. Our results show that no one method is uniformly the best but overall

Identifier	Original		RMCD		NMNC		RMCRO		SRO		MSRO	
	Row	Col	Row	Col	Row	Col	Row	Col	Row	Col	Row	Col
bayer04	304	639	**149**	3641	281	608	630	1037	354	564	183	**282**
bayer09	109	229	43	529	105	219	72	115	52	86	**38**	**57**
bp1600	73	218	248	351	**50**	**183**	224	256	215	238	104	161
extr1	50	98	100	528	43	80	33	54	19	32	**14**	**27**
gre1107	122	321	74	321	119	308	115	166	106	145	**60**	**135**
hydr1	119	260	42	568	93	213	64	104	37	62	**23**	**45**
lhr07c	197	266	94	2646	92	164	111	199	97	171	**56**	**116**
lhr14c	290	372	163	5148	126	**213**	243	429	196	596	**76**	**206**
lns3937	691	1805	151	473	497	1110	80	132	79	131	**61**	**121**
meg1	837	1423	**63**	936	333	937	548	880	281	**471**	329	596
meg4	1653	1794	**167**	2550	1482	1723	1053	1727	1021	1291	512	**838**
nnc1374	40	**78**	123	517	40	78	48	97	**36**	**75**	**37**	**74**
onetone2	236	480	1601	8897	225	446	1711	3147	589	997	**205**	**422**
orani678	473	1341	**102**	**849**	741	1280	889	1273	484	1259	670	**870**
orsreg1	381	756	86	167	381	756	**76**	**150**	**76**	**150**	**76**	**150**
pores3	50	99	34	96	50	99	**12**	19	**12**	19	**12**	**17**
rdist1	75	204	243	515	33	**62**	54	78	45	64	**28**	**62**
sherman3	209	414	94	193	205	401	64	126	**55**	**109**	59	118
wang3	879	1757	646	1288	879	1757	**528**	**1055**	**525**	**1049**	**525**	**1050**
west2021	77	233	**19**	212	67	224	105	167	34	68	**19**	**30**

Table 6.2: The root-mean-square row and column frontsizes ($frow_{rms}$ and $fcol_{rms}$) for the different reordering algorithms. The smallest values of $frow_{rms}$ and $fcol_{rms}$ are highlighted.

the MSRO algorithm gave the best results. We now discuss our findings in more detail.

For almost half the test problems, the RMCD algorithm achieved modest reductions in both the row and column frontsizes. However, while the RMCD algorithm can substantially reduce the row frontsize, it does not directly attempt to control the column frontsize. As a result, the column frontsize can be much larger after reordering than before (for example, problems `bayer09`, `lhr07c`, `lhr14c`, and `hydr1`) and, when used with a frontal solver, more storage and operations may be required for the matrix factorization than are needed for the original ordering. If problem `bayer09` is solved using the frontal solver `MA42` (see Section 7 for details), the factor storage needed for the original ordering is 273 Kwords but for the RMCD ordering it increases to 341 Kwords, and the

number of operations increases from $1.3 * 10^7$ to $2.2 * 10^7$. For some problems, including `nnc1374`, `bp1600`, `extr1`, and `onetone2`, both the row and column frontsizes are worse after reordering with RMCD. There was only one problem, `orani678`, for which the RMCD algorithm gave row and column frontsizes that were smaller than those obtained using the any of the other algorithms.

The RMCRO algorithm also produces inconsistent results. It can do as well as the best of the other algorithms (problems `orsreg1` and `wang3`) but it can also produce an ordering that is worse than the original (problems `bayer04` and `onetone2`). The NMNC algorithm produces much more consistent results. It failed to improve on the original ordering for only one problem (`orani678`). For most problems, the NMNC algorithm is successful in reducing both the row and column frontsizes. However, the reductions are generally less than 30 per cent and, where the RMCRO algorithm performs well, it does better than NMNC.

For a significant proportion of the test examples, the orderings obtained using the SRO algorithm are an improvement on the NMNC orderings. But, in general, MSRO performs much better than both NMNC and SRO: MSRO is only outperformed by SRO for problems `sherman3` and `meg1`. In Table 6.3,

Identifier	Max. entries in a row	Pseudo-diameter
bayer04	33	43
bayer09	34	30
bp1600	304	7
extr1	10	57
gre1107	7	13
hydr1	14	54
lhr07c	63	49
lhr14c	63	41
lns3937	11	34
meg1	411	7
meg4	1194	8
nnc1374	16	17
onetone2	33	23
orani678	1110	6
orsreg1	7	23
pores3	9	22
rdist1	81	54
sherman3	7	30
wang3	7	44
west2021	12	15

Table 6.3: The maximum number of entries in a row and the length of the pseudodiameter of the row graph.

for each test problem we give the maximum number of entries in a row of the matrix together with the length of the pseudodiameter of the row graph. We see that there are four problems, `bp1600`, `meg1`, `meg4`, and `orani678`, that have at least one row with a large number of entries. This in turn results in a short pseudodiamter. The problems with a short pseudodiamter are those for which

the MSRO algorithm performs least well. On the basis of our experiments, we conclude that the MSRO performs well if the pseudodiameter of the associated row graph is sufficiently long and, in general, this will be the case if A has no (nearly) full rows.

6.3 Spectral and hybrid results

We now present results for the spectral and hybrid algorithms. In our experiments involving the spectral method, the Fiedler vector of the row graph was obtained using Chaco 2.0 (Hendrickson and Leland, 1995). We used the multilevel SymmLQ/RQI option and the input parameters were chosen to be the same as those used by Kumfert and Pothen (1997). We again use the weights (W_1, W_2) equal to each of the 13 pairs $(1, 64)$, $(1, 32)$, $(1, 16)$, ..., $(1, 1)$, $(2, 1)$, ..., $(64, 1)$ and select the best result. Our results are presented in Table 6.4. The smallest frontsizes (and those within 5 percent of the smallest) are highlighted in bold. By comparing the results for the spectral ordering with those of the

Identifier	Spectral		Hybrid SRO		Hybrid MSRO		SRO		MSRO	
	Row	Col.	Row	Col.	Row	Col.	Row	Col.	Row	Col.
bayer04	81	151	87	140	**64**	**114**	354	564	183	282
bayer09	27	66	30	60	**24**	**53**	52	86	38	57
bp1600	**51**	175	110	163	**54**	173	215	238	104	**161**
extr1	16	31	16	29	**13**	**26**	19	32	14	27
gre1107	75	135	94	131	**46**	**130**	106	145	60	135
hydr1	17	36	19	32	**12**	**27**	37	62	23	45
lhr07c	68	135	70	139	**46**	**107**	97	171	56	116
lhr14c	88	171	92	132	**77**	**157**	196	596	**76**	206
lns3937	65	128	76	131	**56**	**125**	79	131	61	**121**
meg1	307	591	294	497	**221**	**463**	281	471	329	596
meg4	597	1142	728	1116	**343**	**711**	1021	1291	512	838
nnc1374	33	66	35	70	**27**	**61**	36	75	37	74
onetone2	299	598	313	539	**172**	**379**	589	997	205	422
orani678	**159**	1134	480	1217	499	**694**	484	1240	670	870
orsreg1	82	162	78	154	**77**	154	**76**	**150**	**76**	**150**
pores3	12	18	12	19	**12**	**17**	12	19	**12**	**17**
rdist1	39	**62**	40	**62**	**29**	**62**	45	64	**28**	**62**
sherman3	58	116	**54**	**106**	**53**	**107**	**55**	**109**	59	118
wang3	616	1231	**539**	**1076**	**539**	**1076**	**525**	**1049**	**525**	**1050**
west2021	31	58	24	49	**15**	**28**	34	68	19	30

Table 6.4: The root-mean-square row and column frontsizes ($frow_{rms}$ and $fcol_{rms}$) for the hybrid reordering algorithms. The smallest values of $frow_{rms}$ and $fcol_{rms}$ are highlighted.

original ordering (given in Table 6.2), we see that, in general, applying a spectral method to the row graph of A substantially reduces both the row and column frontsizes. The only problem for which the original ordering was significantly better than the spectral ordering was `onetone2`. This problem is originally well-ordered, with a banded structure, and none of the ordering schemes was able

to improve substantially on the given ordering. Since this problem is already well-ordered for the frontal method, we do not consider it further in this study of row ordering algorithms.

A comparison of the SRO and hybrid SRO results shows that combining a spectral ordering with the SRO method improves the SRO orderings but the hybrid SRO orderings are not consistently better than the spectral orderings. However, the hybrid MSRO orderings are generally better than both the MSRO and the spectral orderings. In particular, for the problems with a short pseudodiameter (see Tables 6.3), the hybrid MSRO algorithm substantially reduces the row frontsizes compared with the MSRO algorithm and, with the exception of `bp1600`, also reduces the column frontsize. If $d(s, e)$ is small, the priority function (5.1) will be insensitive to the W_2 term and the local heuristic of the row and column frontsize gain will largely determine the row ordering. It would appear that the spectral ordering of the interior nodes of the row graph is important and can provide a better guide than the pseudodiameter for the second phase of the MSRO algorithm.

The results presented in this section and the last suggest that, of the reordering algorithms studied in this paper, the hybrid MSRO algorithm yields the best results. For each test problem, this method gave large reductions in both row and column frontsizes. A disadvantage is that the spectral ordering for the row graph must be computed. This calculation can be expensive. We do not want to include detailed timings for the hybrid methods because our codes are written in Fortran while the Chaco package is written in C. Furthermore, the Chaco package performs a large amount of data checking that would not necessarily be required if we were to incorporate code for computing the spectral ordering within our row reordering software package. To give an indication of the time required to compute the spectral ordering, for problem `sherman3`, Chaco took 1.5 seconds and, using the spectral ordering as input data, the hybrid MSRO algorithm required 0.15 seconds. For `lhr14c`, the corresponding timings are 24.8 seconds and 4.9 seconds. Computing the spectral ordering may, therefore, be the most expensive part of the reordering algorithm. If this cost is prohibitive, the MSRO algorithm should be used.

6.4 A comparison with `MC61` orderings

The problems `lns3937`, `meg4`, `nnc1374`, `orsreg1`, `pores3`, `sherman3`, and `wang3`, have sparsity patterns that are symmetric or nearly symmetric (they all have a symmetry index of at least 0.75). As discussed in Section 3, for these problems we can reorder the rows by applying a profile reduction algorithm to the pattern of $A + A^T$. In the Harwell Subroutine Library (1996), routine `MC61` of Reid and Scott (1998) is a profile reduction algorithm that implements a modified version of Sloan's algorithm and includes an option for using a hybrid algorithm. In Tables 6.5 to 6.7 we present results that compare the performance of the MSRO algorithm with that of `MC61` (using default values for all `MC61` control parameters). In Tables 6.5 and 6.6, maximum and root-mean-square frontsizes are given. We see that for each of the problems except `meg4`, the two approaches yield row orderings of comparable quality.

Identifier	Original		MSRO		MC61	
	Row	Col.	Row	Col.	Row	Col.
lns3937	814	2802	77	137	64	146
meg4	2810	3100	1146	1842	248	2671
nnc1374	53	101	53	102	36	105
orsreg1	442	883	102	203	102	204
pores3	75	147	16	22	14	23
sherman3	386	771	102	205	84	173
wang3	901	1801	678	1353	676	1354

Table 6.5: The maximum row and column frontsizes for the MSRO and MC61 reordering algorithms.

Identifier	Original		MSRO		MC61	
	Row	Col.	Row	Col.	Row	Col.
lns3937	691	1805	61	121	54	123
meg4	1653	1794	521	838	162	1380
nnc1374	40	78	37	74	24	70
orsreg1	381	756	76	150	75	149
pores3	50	99	12	17	12	19
sherman3	209	414	59	118	53	108
wang3	879	1757	525	1050	524	1046

Table 6.6: The root-mean-square row and column frontsizes for the MSRO and MC61 reordering algorithms.

For problem meg4, the orderings are quite different, with the MSRO ordering having a smaller column frontsize but a much larger row frontsize compared with that obtained using MC61. If we look at the CPU timings presented in Table 6.7 we observe that MC61 is significantly faster than the MSRO algorithm (for the purposes of comparison, both algorithms are run using two sets of weights (W_1, W_2) since this is the default in MC61). We can account for this by considering the number of edges $|E|$ in the row graph G_R of A used by the MSRO algorithm and in the graph $G(A + A^T)$ used by MC61. For $G(A + A^T)$, if nz is the number of entries in A, $|E| = nz + kz - n$, where $0 \leq kz \ll nz$. G_R is the graph associated with $B = A * A^T$ and, because B is in general much denser than A, the number of edges in G_R is much greater than the number of edges in $G(A + A^T)$. Thus, although both graphs have n nodes (rows), in the row graph each node has many more neighbours. Each time a node is selected for relabelling, the priority of each neighbouring node must be updated. As a result, the MSRO algorithm not only requires more integer storage to hold the row graph, but also takes longer to run than MC61. We therefore recommend that, in the case of matrices with a (almost) symmetric sparsity pattern, MC61 should be used.

Identifier	MSRO		MC61	
	Time	$\|E\|$	Time	$\|E\|$
lns3937	0.39	99041	0.18	24808
meg4	3.34	1942774	0.32	40982
nnc1374	0.12	32376	0.07	8648
orsreg1	0.16	41789	0.10	11928
pores3	0.04	9590	0.02	3700
sherman3	0.24	49899	0.16	15028
wang3	2.43	562894	1.40	151104

Table 6.7: CPU timings and number of edges $|E|$ in the graphs for the MSRO and MC61 reordering algorithms (Sun Ultra 1).

6.5 Adjusting the weights

In this section, we consider the effect of adjusting the weights in the priority function. For his profile reduction algorithm for symmetric matrices, Sloan recommended using the weights $(2, 1)$ but Kumfert and Pothen (1997) found that, for some problems, other values (in particular, $(16, 1)$) gave much better results. We want to consider how sensitive the MSRO and hybrid MSRO ordering algorithms are to the choice of the weights. We have examined the frontsizes 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. In Tables 6.8 and 6.9 we present results for a subset of our test problems. The problems we have selected illustrate the different behaviour we observed. In the tables, percentage increases from the best value of $frow_{rms} * fcol_{rms}$ are given. The percentage increase from the best value of $frow_{rms} * fcol_{rms}$ is also given for the original ordering. We give results for $frow_{rms} * fcol_{rms}$ because the root-mean-square frontsizes are generally less sensitive than the maximum frontsizes to changes in the weights and, although the weights that give the smallest value of $frow_{rms}$ usually also give the smallest value of $fcol_{rms}$, this is not always the case.

We observe that some problems, such as wang3, are relatively insensitive to the choice of weights but in general, the choice of weights makes a significant difference to the performance of the algorithms. However, even a poor choice of weights can give large improvements on the original ordering. For both methods there are problems for which $frow_{rms} * fcol_{rms}$ rises rapidly for large values of W_1/W_2. Following Kumfert and Pothen, we call these class two problems and the rest class one problems. However, we note that a problem may be a class one problem for the MSRO method and a class two problem for the hybrid method. This is illustrated by bayer04. For class one problems, it may be important to choose a large value for W_1/W_2. For class two problems, for the MSRO algorithm, the weights $(1, 1)$ or $(2, 1)$ are generally reasonable choices, while for the hybrid algorithm, $(1, 2)$ is usually a good choice. Since we do not know beforehand to which class a problem belongs, we recommend trying the weights $(2, 1)$ and $(32, 1)$ for the MSRO algorithm and the weights $(1, 2)$ and

Weights	extr1	bayer09	west2021	bayer04	lhr07c	wang3	gre1107
(1,64)	41.3	19.2	187.9	201.3	67.5	0.0	47.9
(1,32)	41.3	17.9	187.9	204.8	66.3	0.0	47.9
(1,16)	41.3	17.6	187.9	173.6	63.1	0.0	47.9
(1,8)	39.1	14.3	120.0	110.7	59.9	0.0	48.3
(1,4)	12.3	1.8	82.4	73.4	64.6	0.0	40.2
(1,2)	1.4	0.0	37.5	72.1	118.5	0.1	20.3
(1,1)	0.0	11.0	25.2	69.8	8.2	0.0	1.5
(2,1)	11.3	35.7	0.0	8.1	5.3	0.1	0.0
(4,1)	67.9	76.5	264.9	6.1	4.3	0.1	1.3
(8,1)	329.7	100.8	794.0	16.3	0.0	0.1	1.3
(16,1)	983.3	97.2	752.9	5.2	0.0	0.1	1.3
(32,1)	983.3	94.1	752.9	0.0	0.0	0.1	1.3
(64,1)	983.3	94.1	752.9	0.0	0.0	0.1	1.3
Original ordering	1179	2024	3052	370	704	180	378

Table 6.8: Percentage increases in $frow_{rms} * fcol_{rms}$ above the minimum value for the MSRO algorithm.

Weights	extr1	bayer09	west2021	bayer04	lhr07c	wang3	gre1107
(1,64)	34.4	37.3	232.3	37.2	77.5	30.2	66.3
(1,32)	26.6	31.8	188.7	24.0	74.7	29.8	61.8
(1,16)	25.8	23.9	116.7	11.1	79.8	29.2	53.4
(1,8)	6.6	11.8	59.1	3.7	66.1	28.0	40.1
(1,4)	2.1	7.2	25.5	0.0	67.7	26.0	24.8
(1,2)	0.0	0.0	14.5	0.4	81.1	22.7	12.4
(1,1)	2.1	8.4	0.0	3.6	94.2	17.7	5.0
(2,1)	2.9	2.4	11.3	43.1	68.7	8.7	0.6
(4,1)	3.6	18.7	13.9	189.2	7.9	0.4	0.0
(8,1)	12.1	13.8	12.8	428.3	7.0	0.0	0.0
(16,1)	38.9	27.3	12.9	429.2	0.0	0.0	0.0
(32,1)	58.0	25.3	12.9	437.6	0.0	0.0	0.0
(64,1)	713.5	25.3	12.9	437.6	0.0	0.0	0.0
Original ordering	1353	1876	4039	3335	960	172	560

Table 6.9: Percentage increases in $frow_{rms} * fcol_{rms}$ above the minimum value for the hybrid MSRO algorithm.

$(32, 1)$ for the hybrid algorithm and, in each case, selecting the better result. In

Identifier	MSRO	Hybrid MSRO
bayer04	0.0	0.4
bayer09	35.7	0.0
bp1600	16.2	7.4
extr1	11.3	0.0
gre1107	0.0	0.0
hydr1	0.0	5.0
lhr07c	0.0	0.0
lhr14c	0.0	16.7
lns3937	0.0	1.6
meg1	42.6	112.0
meg4	5.5	0.0
nnc1374	0.0	0.0
orani678	10.8	84.5
orsreg1	0.2	0.0
pores3	0.0	0.0
rdist11	0.0	12.7
sherman3	0.0	1.5
wang3	0.1	8.8
west2021	0.0	12.9

Table 6.10: Percentage increases in $frow_{rms} * fcol_{rms}$ above the minimum value when the recommended weights are used.

Table 6.10, we show the percentage increases in $frow_{rms} * fcol_{rms}$ from the best value when the recommended weights are used. There are a small number of problems for which the recommended weights give poor results. In particular, problems `meg1` and `orani678` with the hybrid method. As already noted, these problems have a short pseudodiameter (see Table 6.3) and the best results are achieved with a large value of W_2, so that the ordering more closely follows the spectral ordering.

7 Row orderings and frontal solvers

We have developed new algorithms for reordering the rows of unsymmetric matrices for small row and column frontsizes. As discussed in the introduction, the main motivation behind this work is the need for row orderings to improve the efficiency of frontal solvers. In this section, we present results that illustrate the effects of using the MSRO row orderings with a frontal solver.

In the Harwell Subroutine Library (1996), the `MA42` package (Duff and Scott, 1996*b*) is a frontal solver for general unsymmetric problems. The code was primarily designed for unassembled finite-element matrices, but also includes an option for entering the assembled matrix row-by-row, and this is the option we use in our experiments. `MA42` uses reverse communication to obtain information from the user. The structure of the problem is first provided by the user by

calling a subroutine for each row of A. The primary reason for these calls is to establish when variables are fully summed and eligible for elimination. A set of calls to another subroutine enables estimates to be made for the size of the files required to hold the factors and for the maximum row and column frontsizes. In these *symbolic phases*, only the integer indexing information for the rows is used. The numerical factorization is then performed with the user required to call a further subroutine for each row. The information from the earlier symbolic phases is used to control the pivot selection and elimination within the current frontal matrix. Optionally, forward elimination can be performed on a set of right-hand side vectors, in which case a final back-substitution phase yields appropriate solutions. Subsequent right-hand sides can be solved using the matrix factors and a single subroutine call. The code optionally uses direct access files for the matrix factors. This keeps the main memory storage requirements to a minimum. This option is used in our experiments and the `MA42` timings we present in the following tables include the i/o overhead for using direct access files. The "In-core" storage quoted is the storage required for the frontal matrix. In addition, an integer array of length n is required. The "Factor" storage is the sum of the number of real and the number of integer words needed to hold the matrix factors. In our experiments, we use a minimum pivot block size of 16 and we use a version of `MA42` that attempts to exploit zeros within the front (see Scott, 1997 for details).

In our tests with `MA42`, for each problem where values for the entries of the matrix are not supplied, values are generated using the HSL pseudo-random number generator `FA01`. The number of floating-point operations ("ops") counts all operations (+,-,*,/) equally. The "Analyse+Factorize" times include all the time to reorder the matrix, perform the symbolic factorization, and factorize the matrix A. The "Fast Factorize" time is that needed for subsequent factorizations of a matrix with the same sparsity pattern as A. The "Solve" times quoted are for a single right-hand side b and do not include the time required to perform iterative refinement. It should be noted, however, that some of the problems in our test set are so ill-conditioned that iterative refinement is needed for accurate solutions. The experimental results given in Tables 7.1 and 7.2 were obtained on a Sun Ultra 1 and those quoted in Tables 7.3 and 7.4 were obtained on a single processor of a CRAY J932 using 64-bit floating-point arithmetic, and the vendor-supplied BLAS. The CRAY Fortran compiler f90 was used, with default options.

For the problems with a symmetric (or almost symmetric) sparsity pattern, results are given for the original ordering and the `MC61` ordering. For the remaining problems, results are presented for the original ordering, the MSRO ordering, and the hybrid MSRO ordering. In addition, because the RMCD algorithm performed well on problems `meg1` and `orani678`, for these two problems we include results for the RMCD ordering. For the MSRO orderings, the values for the weights recommended in Section 6.5 are used. The timings for the hybrid algorithm do not include the time required to generate the spectral ordering. Consequently, the Analyse+Factorize times for the hybrid algorithm are smaller than those for the MSRO algorithm. The difference between the Analyse+Factorize time and the Fast Factorize time for the original

Identifier	Ordering	Time (seconds)			Factor ops ($*10^6$)	Storage (Kwords)	
		Analyse + Factorize	Fast Factorize	Solve		In-core	Factor
bayer04	Original	22.7	22.1	0.94	288.2	419	3257
	MSRO	18.1	13.3	0.98	269.4	178	3206
	Hybrid	10.1	5.9	0.72	123.8	32	2333
bayer09	Original	1.20	0.96	0.10	13.2	73	273
	MSRO	1.16	0.48	0.07	5.9	16	193
	Hybrid	0.96	0.37	0.06	4.6	6	180
bp1600	Original	0.28	0.20	0.04	2.0	47	61
	MSRO	0.34	0.24	0.03	3.4	81	73
	Hybrid	0.30	0.19	0.02	2.8	54	65
extr1	Original	0.53	0.48	0.06	3.7	14	159
	MSRO	0.54	0.30	0.05	2.7	2	135
	Hybrid	0.52	0.34	0.05	2.5	2	132
gre1107	Original	1.30	1.23	0.08	37.5	84	293
	MSRO	0.64	0.50	0.05	10.4	24	161
	Hybrid	0.71	0.62	0.05	9.9	18	162
hydr1	Original	1.8	1.60	0.13	12.5	57	392
	MSRO	1.3	0.71	0.11	7.6	6	308
	Hybrid	1.1	0.67	0.09	5.5	3	271
lhr07c	Original	12.0	11.7	0.44	125.5	146	1401
	MSRO	8.7	3.8	0.28	57.6	41	1005
	Hybrid	7.7	3.5	0.32	48.7	22	936
lhr14c	Original	24.5	23.9	0.80	235.7	276	2719
	MSRO	19.0	9.2	0.58	149.3	67	2198
	Hybrid	17.0	8.2	0.56	128.8	94	2040
meg1	Original	27.9	27.7	0.50	579.5	2921	1623
	MSRO	14.1	11.5	0.28	273.9	824	1114
	Hybrid	17.2	14.8	0.37	373.9	600	1331
	RMCD	2.4	2.3	0.17	16.6	202	575
orani678	Original	28.9	28.6	0.59	892.2	1605	2284
	MSRO	28.2	11.2	0.26	125.0	2117	808
	Hybrid	40.2	24.2	0.54	1030.0	1263	1883
	RMCD	4.1	3.9	0.22	84.2	368	771
rdist1	Original	4.7	2.6	0.27	90.3	30	1050
	MSRO	2.7	1.2	0.14	15.4	4	405
	Hybrid	2.7	1.3	0.12	17.8	7	429
west2021	Original	0.60	0.45	0.05	3.8	62	131
	MSRO	0.40	0.19	0.04	1.3	3	80
	Hybrid	0.39	0.23	0.04	1.3	2	81

Table 7.1: The results of reordering the rows for the frontal solver MA42 using the MSRO and hybrid MSRO algorithms. (Sun Ultra)

Identifier	Ordered	Time (seconds)			Factor ops ($*10^6$)	Storage (Kwords)	
		Analyse + Factorize	Fast Factorize	Solve		In-core	Factor
lns3937	N	72.1	71.9	1.04	2126.2	2336	3860
	Y	2.2	1.9	0.16	46.6	13	658
meg4	N	41.2	40.8	0.95	1812.4	8769	2208
	Y	6.7	6.1	0.08	3.4	706	219
nnc1374	N	0.54	0.45	0.07	8.1	8	161
	Y	0.50	0.38	0.04	5.9	5	147
orsreg1	N	27.4	27.3	0.56	554.7	410	1538
	Y	2.1	1.9	0.11	33.3	26	395
pores3	N	0.29	0.20	0.02	3.2	14	58
	Y	0.17	0.09	0.01	0.3	1	19
sherman3	N	11.9	11.7	0.47	232.4	313	1201
	Y	2.5	2.2	0.17	33.7	18	477
wang3	N	2137.	2136.	18.9	39972.2	1663	47817
	Y	793.	791.	10.9	14665.8	945	27730

Table 7.2: The results of reordering the rows for the frontal solver MA42 using MC61. (Sun Ultra)

ordering is the time required by MA42 to perform the symbolic factorization. Since the symbolic factorization time is independent of the row order, we can deduce the time taken to reorder the matrix. We see that for the MSRO and hybrid algorithms, the symbolic factorization time is small compared with the reordering and Fast Factorize times. As in Section 6.4, we again see that using MC61 is much less expensive than reordering with an MSRO algorithm. We also observe that it is much more expensive to reorder the matrix on the CRAY. This is because of slow integer arithmetic on the CRAY and means that savings in the total time will only be achieved on the CRAY if a large number of factorizations are to follow the reordering. However, as the ordering routine is separate from the frontal solver, it is possible to reorder the matrix on one machine and then pass the row order to the CRAY for the factorization and solve phases.

The results demonstrate the importance of reordering the rows and illustrate that we are able to achieve substantial savings in the factorization and solve times, the operation count, the in-core storage, and the factor storage. 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 (particularly on the CRAY), the ratio of times, before and after ordering, is not as high as the operation count ratio. Furthermore, MA42 is able to partially offset the effect of a poor ordering by exploiting zeros within the frontal matrix (see also Duff and Scott,

Identifier	Ordering	Time (seconds)		
		Analyse + Factorize	Fast Factorize	Solve
bayer04	Original	11.2	9.8	0.47
	MSRO	36.2	7.9	0.46
	Hybrid	26.3	6.0	0.42
bayer09	Original	1.2	0.99	0.08
	MSRO	3.9	0.74	0.07
	Hybrid	3.3	0.71	0.07
bp1600	Original	0.33	0.27	0.02
	MSRO	0.76	0.26	0.02
	Hybrid	0.59	0.24	0.02
extr1	Original	0.87	0.69	0.04
	MSRO	1.52	0.60	0.03
	Hybrid	1.30	0.58	0.03
gre1107	Original	0.70	0.62	0.02
	MSRO	0.84	0.38	0.01
	Hybrid	0.72	0.37	0.01
hydr1	Original	1.9	1.6	0.06
	MSRO	3.9	1.2	0.06
	Hybrid	2.7	1.1	0.06
lhr07c	Original	4.3	3.6	0.11
	MSRO	33.9	2.7	0.09
	Hybrid	25.3	2.6	0.08
lhr14c	Original	9.1	7.6	0.20
	MSRO	65.9	5.6	0.17
	Hybrid	50.3	5.4	0.16
meg1	Original	9.0	8.7	0.09
	MSRO	19.6	3.3	0.06
	Hybrid	17.5	3.9	0.07
	RMCD	2.2	1.6	0.05
orani678	Original	8.5	8.1	0.13
	MSRO	127.1	3.4	0.06
	Hybrid	113.4	6.9	0.09
	RMCD	2.6	1.8	0.07
rdist1	Original	2.7	2.2	0.06
	MSRO	9.8	1.3	0.04
	Hybrid	8.8	1.3	0.04
west2021	Original	0.71	0.58	0.019
	MSRO	1.37	0.41	0.017
	Hybrid	0.96	0.41	0.017

Table 7.3: The results of reordering the rows for the frontal solver MA42 using the MSRO and the hybrid MSO algorithms. (CRAY J932)

Identifier	Ordered	Time (seconds)		
		Analyse + Factorize	Fast Factorize	Solve
lns3937	N	17.6	17.3	0.21
	Y	2.6	1.3	0.06
meg4	N	15.4	15.0	0.11
	Y	5.2	2.9	0.05
nnc1374	N	0.50	0.41	0.016
	Y	0.75	0.35	0.016
orsreg1	N	5.0	4.9	0.08
	Y	1.5	0.85	0.03
pores3	N	0.20	0.16	0.007
	Y	0.22	0.11	0.005
sherman3	N	3.2	3.5	0.19
	Y	2.6	1.5	0.15
wang3	N	280.3	278.5	2.3
	Y	118.4	109.9	1.4

Table 7.4: The results of reordering the rows for the frontal solver MA42 using MC61. (CRAY J932)

1997 and Cliffe, Duff and Scott, 1998).

8 Comparison with other solvers

In an earlier study, Duff and Scott (1996*a*) compared the performance of MA42 with other codes in the Harwell Subroutine Library designed for the solution of unsymmetric systems of linear equations. Duff and Scott found that, for assembled matrices, no one code was clearly better than the others and that the choice of code depended on the problem being solved. The results also showed that, if the rows of A were poorly ordered, MA42 did not perform well and this highlighted for us the need for a routine to preorder a general sparse matrix for frontal solvers. It is thus of interest to compare the performance of MA42 used with our reordering algorithms with that of other HSL codes. The HSL codes used in our numerical experiments are listed in Table 8.1. For further details, the reader should refer to the given references.

All the HSL codes used in our numerical experiments have control parameters with default values. We use these defaults in each case, even if different codes sometimes choose a different value for essentially the same parameter. The "In-core" storage figures are the minimum amount of main memory required to perform the matrix factorization and solve the linear system $Ax = b$. This figure is the sum of the real and integer storage. We remark that,

if this minimum in-core storage is used, the performance of the codes will often be considerably degraded. We do not give results for matrices with a (almost) symmetric pattern since results for reordering these problems with a profile reduction algorithm applied to $A + A^T$ and then running `MA42` were included in Duff and Scott (1996*a*). Based on the results presented in Table 7.1, we reorder the matrix `meg1` using the RMCD algorithm, for problem `rdist1` we use the MSRO algorithm, and for the other examples we use the hybrid MSRO algorithm.

Code	Description
`MA38`	Implements a combined unifrontal /multifrontal algorithm. Uses approximate minimum degree ordering. (Davis and Duff 1993)
`MA41`	Implements a multifrontal algorithm. Option to preorder matrix for zero-free diagonal. Uses approximate minimum degree ordering. (Amestoy and Duff 1989)
`MA48`	General sparse unsymmetric solver. Uses Markowitz criteria. (Duff and Reid 1993, Duff and Reid 1996)

Table 8.1: HSL sparse unsymmetric linear equation solvers used in our numerical experiments.

Our results are presented in Tables 8.2 (Sun) and 8.3 (CRAY). We see that, while `MA42` rarely has the smallest factorization time, for many of the test cases at least one, and sometimes two, of the other codes is less competitive in terms of factorization times, operation counts, and factor storage. Moreover, since `MA42` is currently the only code to allow the factors to be held in auxillary storage, `MA42` is clearly the code of choice if minimizing the use of main memory is the prime concern. The `MA42` Solve times are greater than for the other solvers even if the factors have a similar number of entries because of the i/o overhead. If the user is able to hold the factors in-core, the `MA42` Solve times are competitive (see Duff and Scott, 1996*a*).

Identifier	Code	Time (seconds)			Factor ops	Storage (Kwords)	
		Analyse + Factorize	Fast Factorize	Solve	($*10^6$)	In-core	Factor
bayer04	MA42	10.1	5.9	0.72	123.8	32	2333
	MA41†	14.9	9.5	0.45	159.0	2757	1928
	MA48	7.8	1.3	0.13	15.7	1130	926
	MA38	11.9	6.3	0.24	55.7	1557	1138
bayer09	MA42	0.96	0.37	0.06	4.6	6	180
	MA41†	0.90	0.54	0.05	6.6	304	179
	MA48	0.46	0.15	0.02	2.6	135	112
	MA38	0.79	0.35	0.02	1.9	169	110
bp1600	MA42	0.31	0.20	0.02	2.9	54	66
	MA41†	0.16	0.09	0.01	0.72	67	35
	MA48	0.04	0.01	0.01	0.02	29	12
	MA38	0.09	0.04	0.01	0.03	27	17
extr1	MA42	0.55	0.32	0.06	2.53	2	132
	MA41†	0.38	0.21	0.03	1.17	175	92
	MA48	0.24	0.01	0.01	0.24	81	55
	MA38	0.50	0.23	0.02	0.50	136	82
gre1107	MA42	0.72	0.61	0.05	9.9	18	162
	MA41	1.12	1.05	0.04	20.5	282	192
	MA48	0.68	0.35	0.01	6.5	144	131
	MA38	0.74	0.53	0.02	6.2	161	101
hydr1	MA42	0.89	0.63	0.11	5.47	3	271
	MA41†	1.01	0.64	0.08	5.59	408	243
	MA48	0.80	0.14	0.02	1.46	2134	167
	MA38	1.23	0.60	0.04	2.40	253	194
lhr07c	MA42	8.0	3.5	0.30	48.7	22	936
	MA41†	11.2	9.7	0.28	151.6	2416	1387
	MA48	11.8	4.0	0.11	56.3	1553	1253
	MA38	9.1	5.6	0.19	33.9	1317	936
lhr14c	MA42	17.7	8.9	0.62	129.1	94	2041
	MA41†	48.1	44.1	0.71	315.2	5025	3498
	MA48	24.4	7.0	0.22	88.8	2978	2399
	MA38	16.6	10.2	0.35	63.5	2201	1728
meg1	MA42	2.4	2.3	0.17	16.6	202	575
	MA41†	5.4	4.9	0.14	145.0	1203	702
	MA48	1.8	0.4	0.02	4.3	368	282
	MA38	2.0	1.2	0.05	10.6	472	328
rdist1	MA42	2.7	1.2	0.14	15.4	4	405
	MA41†	1.7	1.1	0.11	9.67	598	323
	MA48	7.9	1.7	0.08	25.3	1001	812
	MA38	3.3	2.4	0.09	22.4	752	503
west2021	MA42	0.32	0.22	0.05	1.32	2	82
	MA41†	0.19	0.09	0.02	0.35	104	47
	MA48	0.10	0.01	0.01	0.05	44	25
	MA38	0.24	0.22	0.01	0.06	73	43

Table 8.2: A comparison of HSL codes on unsymmetric assembled problems. † denotes the matrix is first preordered to have a zero-free diagonal. (Sun Ultra)

Identifier	Code	Time (seconds)		
		Analyse + Factorize	Fast Factorize	Solve
bayer04	MA42	26.3	6.0	0.42
	MA41†	24.8	8.0	0.25
	MA48	20.2	4.3	0.11
	MA38	16.1	4.1	0.21
bayer09	MA42	3.3	0.71	0.07
	MA41†	2.0	0.71	0.03
	MA48	1.5	0.30	0.01
	MA38	1.7	0.44	0.03
bp1600	MA42	0.30	0.21	0.01
	MA41†	0.41	0.12	0.01
	MA48	0.16	0.01	0.01
	MA38	0.22	0.07	0.01
extr1	MA42	1.30	0.58	0.032
	MA41†	0.99	0.37	0.032
	MA48	0.95	0.08	0.015
	MA38	1.35	0.38	0.028
gre1107	MA42	0.73	0.36	0.013
	MA41	0.93	0.61	0.013
	MA48	1.16	0.18	0.007
	MA38	0.86	0.22	0.014
hydr1	MA42	2.7	1.13	0.06
	MA41†	2.5	0.91	0.06
	MA48	2.6	0.26	0.03
	MA38	3.0	0.76	0.05
lhr07c	MA42	24.8	2.5	0.08
	MA41†	11.2	6.0	0.08
	MA48	21.5	2.4	0.06
	MA38	10.1	3.0	0.09
lhr14c	MA42	49.5	5.3	0.16
	MA41†	32.5	16.6	0.17
	MA48	46.2	4.8	0.11
	MA38	20.6	5.9	0.17
meg1	MA42	2.2	1.6	0.05
	MA41†	5.9	3.5	0.04
	MA48	3.9	1.2	0.02
	MA38	2.6	1.1	0.03
rdist1	MA42	9.8	1.3	0.04
	MA41†	3.1	1.1	0.06
	MA48	15.3	1.7	0.03
	MA38	4.1	1.3	0.04
west2021	MA42	0.96	0.41	0.02
	MA41†	0.59	0.22	0.03
	MA48	0.46	0.03	0.01
	MA38	0.85	0.27	0.02

Table 8.3: A comparison of HSL codes on unsymmetric assembled problems. † denotes the matrix is first preordered to have a zero-free diagonal. (CRAY J932)

9 Conclusions

In this paper, we have looked at the problem of reordering the rows of a general unsymmetric matrix A for use with a frontal solver. We have used the row graph of A and applied variants of Sloan's algorithm to this graph. We have found that the SRO algorithm that applies Sloan's algorithm directly to the row graph does not perform as consistently well as the MSRO algorithm, which uses a modified priority function that attempts to directly limit the growth in the row and column frontsizes at each assmbly step. The MSRO algorithm works well on a wide range of problems and in general produces orderings that are much better than those obtained by the existing RMCD and NMNC ordering algorithms. The only problems we have found that it does not work well on are those for which the row graph has a short pseudodiameter. For problems with an almost symmetric sparsity pattern, for efficiency reasons, we recommend using a profile reduction algorithm to reorder the pattern of $A + A^T$.

We have also looked at applying the spectral method to the row graph. Our results suggest that the hybrid MSRO method is superior to the spectral method and generally out-performs MSRO. However, 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 the pseudodiameter that provides start and end nodes for the MSRO algorithm. 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 MSRO algorithm may be preferred. In our experiments with the frontal method, we observed that the cost of computing a row ordering can significantly increase the time taken to perform the analyse phase, but the savings in the factorization and solve times, as well as in the main memory requirements and factor storage, achieved through using an efficient ordering can justify the ordering cost, particularly if more than one matrix with the same sparsity pattern is to be factorized and the computer used has fast integer arthimetic. An interesting question is whether it is possible to design algorithms that compute orderings with the same quality as the MSRO algorithms but at a significantly lower cost.

10 Acknowledgements

I would like to thank Kyle Camarda for sending me a copy of his code that implements the NMNC algorithm. This code was used to obtain the NMNC results included in Table 6.2. I am also grateful to my colleagues Iain Duff and John Reid for their interest in this work and for commenting on a draft of this paper.

References

P.R. Amestoy and I.S. Duff. Vectorization of a multiprocessor multifrontal code. *Inter. Journal of Supercomputer Applics*, **3**, 41–59, 1989.

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.

K.V. Camarda. *Ordering strategies for sparse matrices in chemical process simulation.* PhD thesis, University of California, 1997.

K.A. Cliffe, I.S. Duff, and J.A. Scott. Performance issues for frontal schemes on a cache-based high performance computer. *Inter. Journal on Numerical Methods in Engineering*, **42**, 127–143, 1998.

A.B. Coon. *Orderings and direct methods for coarse granular parallel solutions in equation-based flowsheeting.* PhD thesis, University of Illinois, 1990.

A.B. Coon and M.A. Stadtherr. Generalized block-triangular matrix orderings for parallel computation in process flowsheeting. *Comput. Chem. Engng.*, **96**, 787–805, 1995.

E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. *in* 'Proceedings of the 24th National Conference of the ACM'. Brandon Systems Press, 1969.

T. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, **97**(23), 1997.

T.A. Davis and I.S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. Technical Report RAL-93-036, Rutherford Appleton Laboratory, 1993.

J.J. Dongarra, J. DuCroz, I.S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Mathematical Software*, **16**(1), 1–17, 1990.

I.S. Duff. MA32 - a package for solving sparse unsymmetric systems using the frontal method. Report AERE R10079, Her Majesty's Stationery Office, London, 1981.

I.S. Duff. Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *SIAM J. Scientific and Statistical Computing*, **5**, 270–280, 1984.

I.S. Duff and J.K. Reid. MA48, a Fortran code for direct solution of sparse unsymmetric linear systems of equations. Report RAL-93-072, Rutherford Appleton Laboratory, 1993.

I.S. Duff and J.K. Reid. The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. Mathematical Software*, **22**, 187–226, 1996.

I.S. Duff and J.A. Scott. A comparison of frontal software with other sparse direct solvers. Technical Report RAL-TR-96-102 (Revised), Rutherford Appleton Laboratory, 1996*a*.

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*b*.

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, 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.

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, 1996.

E. Hellerman and D. Rarick. The partitioned preassigned pivot procedure (P^4). *in* D. Rose and R. Willoughby, eds, 'Sparse Matrices and their and Applications', pp. 67–76. Plenum Press, 1972.

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

P. Hood. Frontal solution program for unsymmetric matrices. *Inter. Journal on Numerical Methods in Engineering*, **10**, 379–400, 1976.

B.M. Irons. A frontal solution program for finite-element analysis. *Inter. Journal on Numerical Methods in Engineering*, **2**, 5–32, 1970.

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

J.W.H. Liu and A.H. Sherman. Comparative analysis of the Cuthill-McKee and the Reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM J. Numerical Analysis*, **13**, 198–213, 1976.

B. H. Mayoh. A graph technique for inverting certain matrices. *Mathematics of Computation*, **19**, 644–646, 1965.

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.

J.A. Scott. Element resequencing for use with a multiple front algorithm. *Inter. Journal on Numerical Methods in Engineering*, **39**, 3999–4020, 1996.

J.A. Scott. Exploiting zeros in frontal solvers. Technical Report RAL-TR-98-041, Rutherford Appleton Laboratory, 1997.

J.A. Scott. On ordering elements for a frontal solver. Technical Report RAL-TR-98-031, Rutherford Appleton Laboratory, 1998. To appear in *Communications in Numerical Methods in Engineering.*

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.

M.A. Stadtherr and J.A. Vegeais. Process flowsheeting on supercomputers. *IChemE Symp. Ser.*, **92**, 67–77, 1985.