NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl., **6,** 189–211 (1999)

A New Row Ordering Strategy for Frontal Solvers

Jennifer A. Scott*

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

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 are interested in matrices that have a highly asymmetric structure. 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 applying the spectral method to the row graph. Numerical experiments performed on a range of practical problems illustrate that our proposed MSRO and hybrid MSRO row ordering algorithms yield substantial reductions in the front sizes and, when used with a frontal solver, significantly enhance its performance both in terms of the factorization time and storage requirements.

KEY WORDS ordering rows; frontal method; row graphs; sparse unsymmetric matrices

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 [15,16], it can be used to solve any general linear system of equations [8,9]. In this paper, we are concerned with using the frontal method for unsymmetric non-element problems; in a separate paper [24] we discuss ordering strategies for element problems.

* Correspondence to J. A. Scott, Department for Computation and Information, Atlas Centre, Rutherford Appleton Laboratory, Oxon OX11 0QX, UK.

CCC 1070–5325/99/030189–23 $17.50

Received 19 September 1998
Revised 25 February 1999

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 front size* and the number of columns the *column front size*. 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 front sizes 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) [7] 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 front sizes as small as possible. If $frow_i$ and $fcol_i$ denote the row and column front sizes before the ith elimination, we are interested in

- the maximum row and column front sizes

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

since these determine the amount of main memory needed,

- the root-mean-square row and column front sizes

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

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

- the average size of the frontal matrix

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

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,

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

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 front sizes 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 front sizes will be large. Clearly, the frontal method is not a good choice for such systems.

Throughout this study, we shall be concerned with matrices that have a highly asymmetric structure. For matrices with a symmetric structure, the rows can be successfully ordered using a profile reduction algorithm such as that of Sloan [25] and Reid and Scott [20]. For matrices with an almost symmetric pattern, good orderings can generally be obtained by applying a profile reduction algorithm to the sparsity pattern of $A + A^{\mathrm{T}}$. Results illustrating this are given in [23]. We shall also only consider orderings for use with a 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 [2] and, for element problems, by Scott [21], 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 also explain the Cuthill–McKee and Sloan algorithms for reordering the nodes of an undirected graph. Our new reordering algorithms are introduced in Section 3. 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 Sections 4 and 5. In Section 6, we use the new algorithms with the frontal solver MA42. Finally, some concluding remarks are made in Section 7.

2. Graphs and matrix reordering

2.1. Basic definitions

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*. In our discussion of row ordering algorithms, we will assume that the graph G is connected. If not, it is straightforward to apply the algorithms to each component of G.

We now establish the well-known 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 non-zero. 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.

For unsymmetric matrices, one possibility for developing row ordering algorithms is to use a bipartite graph. 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. The bipartite graph has been used with some success by, for example, Coon [4] and Coon and Stadtherr [5]. However, reordering techniques for undirected graphs have been the subject of much research and, if possible, we would like to be able to exploit some of these techniques. This motivates us to consider in this paper using row graphs. Row graphs were first introduced by Mayoh [19] and were used to permute matrices to singly bordered block diagonal form. 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 non-zero 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 non-zero. Row permutations of A correspond to relabelling the nodes of the row graph.

From our discussion in Section 1, it is clear that for the frontal method to be efficient the rows of A should be ordered so that the matrix has a variable band form, with the band as narrow as possible. Since we plan to exploit ordering techniques for undirected graphs, in the following subsections we briefly outline two algorithms for bandwidth and profile reduction of symmetric matrices, namely, the reverse Cuthill–McKee algorithm and the

Sloan algorithm. In Section 3, we consider using these algorithms to reorder the nodes of the row graph G_R to produce row orderings that are appropriate for use with a frontal solver.

2.2. *The reverse Cuthill–McKee algorithm*

The reverse Cuthill–McKee algorithm is primarily aimed at reducing the *bandwidth* B of a symmetrically structured matrix where

$$B = \max_{(i,j)\in E} (i + 1 - j) \tag{2.1}$$

The algorithm divides the nodes $V = \{1, 2, ..., n\}$ 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 [18], where the *profile* P is defined to be

$$P = \sum_{i=1}^{n} \max_j \{i + 1 - j : (i, j) \in E\} \tag{2.2}$$

The root r of the level structure is usually chosen to one of the ends of a pseudo-diameter 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,

$$D(G) = \max\{d(i, j) : i, j \in V\} \tag{2.3}$$

A *pseudo-diameter* $\delta(G)$ is defined by any pair of nodes i and j in G for which $d(i, j)$ is close to $D(G)$. A pseudo-diameter may be found efficiently using a modified version of the Gibbs–Poole–Stockmeyer algorithm [13] (see [20,26]).

2.3. *The Sloan algorithm*

The Sloan algorithm is designed to reduce the profile (2.2) of a symmetrically structured matrix. It has two distinct phases:

1. selection of a start node and a target end node;
2. node reordering.

In phase 1, a pseudo-diameter of G is computed. One end s of the pseudo-diameter 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 pseudo-diameter 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

$$P_i = -W_1 cdeg_i + W_2 d(i, e) \tag{2.4}$$

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 with 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).

3. New row ordering algorithms

We now address the problem we are interested in, that is, the reordering of the rows of an unsymmetric matrix A for use with a frontal solver.

3.1. The RCMRO algorithm

As mentioned earlier, if the matrix A has a variable band structure, the row and column front sizes 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.

3.2. 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 front size.

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.

3.2.1. Example

We now illustrate the SRO method using the matrix with the sparsity pattern given in Figure 1. The neighbours of each row are listed in Table 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 1, the lifetimes are given in Table 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 $(4, 6)$ $(d(4, 6) = 3$ and $d(i, j) \leq 3, i, j = 1, 2, ..., 6)$, and the initial priorities are then computed (Table 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 are updated, resulting in the matrix of Figure 2 (rows with the priority given as—have been reordered). 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 which they are assembled is arbitrary. Assuming row 5 is ordered first, we obtain the final reordered matrix given in Figure 4. The sum of the lifetimes for the reordered matrix is 18.

	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 1. The original matrix.

Table 1. Lists of neighbours and initial priorities for SRO method

Row i	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

	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 2. Partially ordered matrix

	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 3. Partially ordered matrix

	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 4. Final reordered matrix

3.3. 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 front sizes, 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

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

Here $rcgain_i = rgain_i + cgain_i$, where $rgain_i$ and $cgain_i$ are the increases to the row and column front sizes resulting from assembling row i next. Assembling a row into the frontal matrix causes the row front size to either increase by one, to remain the same, or to decrease. The row front size 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 front size 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 front size decreases. Hence, if s_i is the number of columns that become fully summed when row i is assembled,

$$rgain_i = 1 - s_i \tag{3.2}$$

and

$$cgain_i = newc_i - s_i \tag{3.3}$$

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

$$rcgain_i = 1 + newc_i - 2s_i \tag{3.4}$$

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.

3.3.1. Example

We now illustrate the MSRO method, again using the matrix given in Figure 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 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.

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 6.

Table 2. Initial priorities for MSRO method

Row i	$rcgain_i$	$d(i,6)$	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

	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. 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 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 7. Final reordered matrix

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 7. The sum of the lifetimes for the reordered matrix is 16.

3.4. Spectral ordering algorithms

Spectral algorithms have been used in recent years for matrix profile and wavefront reduction. Barnard, Pothen and Simon [1] described a spectral algorithm that associates a Laplacian matrix L with a given matrix $S = \{s_{ij}\}$ with a symmetric sparsity pattern,

$$L = \{l_{ij}\} = \begin{cases} -1 & \text{if } i \neq j \text{ and } \quad s_{ij} \neq 0 \\ 0 & \text{if } i \neq j \text{ and } \quad s_{ij} = 0 \\ \sum_{i \neq j} |l_{ij}| & \text{if } i = j. \end{cases} \tag{3.5}$$

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 non-increasing or non-decreasing order.

For matrices A with an unsymmetric sparsity pattern, we can apply the spectral method to the symmetric matrix $B = A * A^{\mathrm{T}}$, 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 4), we call this method the *spectral row reordering* algorithm.

3.5. A hybrid method

When ordering symmetrically structured matrices for a small profile, Kumfert and Pothen [17] observed that spectral orderings do well in a global sense but are often poor locally. They therefore proposed 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 pseudo-diameter. Further experiments by Reid and Scott [20,24] 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 (2.4), Reid and Scott used the priority function

$$P_i = -W_1 cdeg_i - W_2(h/n)p_i \tag{3.6}$$

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 (3.1). 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 $B = A * A^{\mathrm{T}}$. The first will then use the priority function (3.6) and the second will generalise (3.1) and use the priority function

$$P_i = -W_1 rcgain_i - W_2(h/n)p_i \tag{3.7}$$

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. In future work we plan to look at using other input orderings.

4. 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.

Table 3. The test problems

Identifier	Order	Number of entries	Description/discipline
bayer04	20 545	159 082	Chemical process simulation
bayer09	3 083	21 216	Chemical process simulation
bp1600	1 600	4 841	Basis matrix from LP problem
extr1	2 837	11 407	Dynamic simulation problem
gre1107	1 107	5 664	Simulation studies in computer systems
hydr1	5 308	23 752	Dynamic simulation problem
lhr07c	7 337	156 508	Light hydrocarbon recovery
lhr14c	14 270	307 858	Light hydrocarbon recovery
meg1	2 904	58 142	Chemical process simulation
onetone2	36 057	227 628	Harmonic balance method, one-tone
orani678	2 529	90 158	Economic model
rdist1	4 134	94 408	Reactive distillation problem
west2021	2 021	7 353	Chemical engineering

4.1. 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 3. The problems are all taken from either the Harwell–Boeing collection [10] or the University of Florida Sparse Matrix Collection [6]. Note that all the test matrices have a highly asymmetric structure, with a symmetry index of less than 0.2.

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. In the experiments involving the spectral method, the Fielder vector of the row graph is obtained using Chaco 2.0 [27].

4.2. Performance of the new algorithms

We first compare the performance of the new row ordering algorithms that were introduced in Section 3. For the SRO and MSRO algorithms (and hybrid versions), 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 (we illustrate the sensitivity of the algorithms to the choice of the weights in the next subsection). In Table 4, the average front size $f_{\text{avg}} * 10^2$ is given. We highlight in bold the smallest values for each problem (and any that are within five per cent of the smallest). For comparison, we include the original ordering (although it should be noted that the original ordering may not be particularly significant, since this is just the order in which the data were supplied when they were included in the sparse matrix test set). Our results show that, in general, the smallest average front size is obtained using the hybrid MSRO algorithm. We now discuss our findings in more detail.

For a significant proportion of the test examples, the orderings obtained using the RCMRO algorithm are a significant improvement on the original ordering. But, in each case, the SRO ordering is better than the RCMRO ordering and, in general, MSRO produces an ordering

Table 4. The average front size ($f_{\text{avg}} * 10^2$) for the new row ordering algorithms. The smallest values are highlighted

Identifier	Original	RCMRO	SRO	MSRO	Spectral	Hybrid SRO	Hybrid MSRO
bayer04	1 909	5 505	1 993	515	120	122	**72**
bayer09	248	65	43	21	14	17	**12**
bp1600	147	474	522	157	**86**	178	90
extr1	49	17	6	4	4	5	**3**
gre1107	386	190	153	81	100	122	**59**
hydr1	310	64	23	10	6	6	**3**
lhr07c	521	218	166	62	90	95	**48**
lhr14c	1 076	1 030	717	153	145	154	**117**
meg1	11 823	4 687	1 298	1 949	1 784	1 438	**1 014**
onetone2	1 131	51 041	5 843	864	1 782	1 683	**648**
orani678	6 193	9 719	5 431	5 828	**1 659**	5 049	2 265
rdist1	146	42	29	**17**	23	25	**17**
west2021	179	172	20	6	16	11	**4**

Table 5. The maximum number of entries in a row and the length of the pseudo diameter of the row graph

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
meg1	411	7
onetone2	33	23
orani678	1 110	6
rdist1	81	54
west2021	12	15

with a smaller average front size than SRO: MSRO is only outperformed by SRO for problems orani678 and meg1. This shows that, although worthwhile reductions in the front sizes can be achieved by applying an existing bandwidth or profile reduction algorithm to the row graph of A, the more sophisticated approach that modifies the priority function in an attempt to limit both the row and column front sizes has greater success.

By comparing the results for the spectral ordering with those of the original ordering, we see that, in general, applying a spectral method to the row graph of A also substantially reduces the average front size. A comparison of the SRO and hybrid SRO results shows that combining a spectral ordering with the SRO method improves the SRO ordering but the hybrid SRO ordering is often poorer than the spectral ordering. However, the hybrid MSRO orderings outperform both the MSRO and the spectral orderings.

To try and get an insight into when the MSRO and hybrid MSRO algorithms perform well, we consider the pseudo-diameters of the matrices. In Table 5, for each test problem we give the maximum number of entries in a row of the matrix together with the length of the pseudo-diameter of the row graph. We see that there are three problems, bp1600, meg1 and orani678, that have at least one row with a large number of entries. This in turn results in a short pseudo-diameter. The problems with a short pseudo-diameter 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 pseudo-diameter of the associated row graph is sufficiently long and, in general, this will be the case if A has no rows with a large proportion of non-zero entries.

For the problems with a short pseudo-diameter, the hybrid MSRO algorithm substantially reduces the average front size compared with the MSRO algorithm. If $d(s, e)$ is small, the priority function (3.1) will be insensitive to the W_2 term and the local heuristic of the row and column front size 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 pseudo-diameter for the second phase of the MSRO algorithm.

The results presented in this section suggest that, of the new row ordering algorithms introduced in this paper, the hybrid MSRO algorithm yields the best results. For each test

problem, this method gives large reductions in the average front size. A disadvantage is that the spectral ordering for the row graph must be computed. This calculation can be relatively expensive. We do not 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 `lhr14c`, Chaco took 25 seconds and, using the spectral ordering as input data, the hybrid MSRO algorithm required 5 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. The importance of the cost of reordering the rows is discussed further in Section 6.

4.3. Adjusting the weights

In this section, we consider the effect of adjusting the weights in the priority function for the MSRO and hybrid MSRO algorithms. For his profile reduction algorithm for symmetric matrices, Sloan [25] recommended using the weights (2, 1) but Kumfert and Pothen [17] 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 front sizes 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 and 7 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 f_{avg} are given.

We observe that the choice of weights can make 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 f_{avg} rises rapidly for large values of W_1/W_2. Following Kumfert and Pothen [17], we call these class 2 problems

Table 6. Percentage increases in average front size above the minimum value for the MSRO algorithm

Weights	`orani678`	`extr1`	`bayer09`	`west2021`	`bayer04`	`lhr07c`	`gre1107`
(1, 64)	10.0	41.6	18.9	186.9	202.3	72.1	47.7
(1, 32)	12.6	41.6	17.5	186.9	205.2	70.8	47.7
(1, 16)	6.8	41.6	17.2	186.9	174.6	67.7	47.7
(1, 8)	3.9	39.5	13.9	118.8	110.8	68.1	48.1
(1, 4)	4.4	12.6	1.5	81.1	73.7	64.6	40.0
(1, 2)	0.0	1.4	0.0	35.9	72.5	122.8	20.6
(1, 1)	4.7	0.0	10.9	24.7	69.9	9.3	1.6
(2, 1)	10.7	11.1	35.4	0.0	8.2	7.0	0.0
(4, 1)	10.7	66.9	76.7	266.4	6.5	5.8	1.5
(8, 1)	10.7	329.0	101.7	795.9	16.5	0.0	1.5
(16, 1)	10.7	918.8	97.5	756.0	5.3	0.0	1.5
(32, 1)	10.7	918.8	94.2	756.0	0.0	0.0	1.5
(64, 1)	10.7	918.8	94.2	756.0	0.0	0.0	1.5

Table 7. Percentage increases in average front size above the minimum value for the hybrid MSRO algorithm

Weights	orani678	extr1	bayer09	west2021	bayer04	lhr07c	gre1107
(1, 64)	0.0	34.0	37.6	205.8	36.9	81.5	66.1
(1, 32)	67.2	26.3	32.7	166.7	23.6	78.6	61.6
(1, 16)	73.7	15.7	24.1	102.0	10.7	74.6	53.2
(1, 8)	76.0	6.3	12.0	50.9	3.4	69.7	4001
(1, 4)	41.2	1.9	7.3	21.0	0.0	70.8	24.9
(1, 2)	91.0	0.0	0.0	12.5	0.5	84.1	12.6
(1, 1)	227.1	2.1	9.1	0.0	4.1	96.2	5.3
(2, 1)	373.0	2.7	3.6	11.6	43.1	70.7	0.7
(4, 1)	173.8	3.4	21.0	14.2	191.2	7.6	0.0
(8, 1)	173.8	11.8	15.0	13.1	426.3	6.6	0.0
(16, 1)	173.8	38.7	29.2	13.3	427.2	0.0	0.0
(32, 1)	173.8	58.4	27.1	13.3	435.8	0.0	0.0
(64, 1)	173.8	712.0	27.1	13.3	435.8	0.0	0.0

Table 8. Percentage increases in average front size above the minimum value when the recommended weights are used

Identifier	MSRO	Hybrid MSRO
bayer04	0.0	0.5
bayer09	46.4	0.0
bp1600	18.0	1.2
extr1	11.3	0.0
gre1107	0.0	0.0
hydr1	0.0	4.7
lhr07c	0.0	0.0
lhr14c	0.0	17.5
meg1	43.9	9.6
onetone2	0.0	0.0
orani678	10.7	0.0
rdist1	0.0	12.2

and the rest class 1 problems. However, we note that a problem may be a class 1 problem for the MSRO method and a class 2 problem for the hybrid method. This is illustrated by bayer04. For class 1 problems, it may be important to choose a large value for W_1/W_2. For class 2 problems, for the MSRO algorithm, the weights (1, 1) or (2, 1) are generally reasonable choices. Since we do not know beforehand to which class a problem belongs, for the MSRO algorithm we recommend trying the weights (2, 1) and (32, 1) and selecting the better result. For the hybrid algorithm, we recommend the weights (1, 2) and (32, 1), unless the matrix has a short pseudo-diameter. In this case, the best results are achieved with a large value of W_2, so that the ordering more closely follows the spectral ordering. We thus make a further recommendation that, if the row graph has a short pseudo-diameter, the hybrid method should use the weights (1, 64). In Table 8, we show the percentage increases in f_{avg} from the best value when the recommended weights are used.

5. Comparisons with other row ordering algorithms

So far, we have presented results for the new row ordering algorithms introduced in this paper and, on the basis of these results, we recommend the MSRO or hybrid MSRO algorithms. In this section, we compare the performance of these new algorithms with two existing row ordering algorithms that are designed for unsymmetric matrices for use with frontal solvers.

The restricted minimum column degree (RMCD) algorithm was recently discussed by Camarda [2]. 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 non-zero. 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 non-zero 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.

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 [4] and Coon and Stadtherr [5] and modified by Camarda [2]. 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 minimized. The new minimum net cut (NMNC) algorithm of Camarda is more expensive to implement than the simple RMCD algorithm but the results presented in [2] show that it can yield better orderings.

We have performed numerical experiments using both the RMCD and NMNC algorithms. Our findings are presented in Table 9. We see 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. In none of the test examples did NMNC produce the smallest average front size. In most cases, the MSRO and hybrid MSRO algorithms perform much better than RCMD: there are only three problems, `bp1600`, `meg1` and `orani678`, for which RMCD gives the best results. These are the problems that have small pseudo-diameters (see Table 5), and we have already seen that the MSRO algorithms perform relatively poorly on these problems.

6. Row orderings and frontal solvers

We have developed new algorithms for reordering the rows of unsymmetric matrices for small front sizes. 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 [14], the `MA42` package of Duff and Scott [11] 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

Table 9. The average front size ($f_{avg} * 10^2$) for the different reordering algorithms. The smallest values are highlighted

Identifier	Original	RMCD	NMNC	MSRO	Hybrid MSRO
bayer04	1 909	4 162	1 682	515	**72**
bayer09	248	152	229	21	**12**
bp1600	147	**56**	86	157	90
extr1	49	486	34	4	**3**
gre1107	386	124	364	81	**59**
hydr1	311	231	197	10	**3**
lhr07c	521	2 180	150	62	**48**
lhr14c	1 076	7 645	266	153	**117**
meg1	11 823	**461**	3 094	1 949	1 014
onetone2	1 131	141 510	1 001	864	**648**
orani678	6 193	**616**	8 959	5 828	2 265
rdist1	146	1 252	20	**17**	**17**
west2021	179	28	151	6	**4**

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 front sizes. 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 input/output 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 [22] 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. The experimental results given in Table 10 were obtained on a Sun Ultra 1, and those quoted in Table 11 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.

Results are presented for the original ordering, the MSRO ordering, and the hybrid MSRO ordering, using the values for the weights recommended in Section 4.3. In addition, because the RMCD algorithm performed well on problems `bp1600`, `meg1` and `orani678`, for these problems we include results for the RMCD ordering. 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 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 MSRO algorithms, the symbolic factorization time is small compared with the reordering and 'Fast Factorize' times. We observe that it is much more expensive to reorder the matrix on the CRAY. This is because of slow integer arithmetic on the CRAY. However, as the ordering routine is separate from the frontal solver, if the reordering time is important to the user, the matrix can be reordered on one machine and the row order then passed to the CRAY for the factorization and solve phases. In many practical applications, the number of factorizations of matrices with the same structure but different numerical values is likely to be large. In this case, the cost of a single matrix reordering will be an insignificant part of the total cost and it is worthwhile spending extra time getting an improved ordering.

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 [3,12]).

7. 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 reverse Cuthill–McKee and 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 outperforms Cuthill–McKee but generally does not perform as well as the MSRO algorithm, which uses a modified priority function that attempts to directly limit the growth in the row and column front sizes at each assembly 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 on which it works less well are those for which the row graph has a short pseudo-diameter.

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 outperforms MSRO. A possible disadvantage of the hybrid method is the need to compute a global priority function. The time taken to compute a spectral ordering is greater than that needed to compute the

Table 10. The results of reordering the rows for the frontal solver MA42 using the MSRO and hybrid MSRO algorithms. (Sun Ultra)

		CPU time (seconds)				Storage (Kwords)	
Identifier	Ordering	Analyse + Factorize	Fast Factorize	Solve	Factor ops ($\times 10^6$)	In core	Factor
bayer04	Original	22.7	22.1	0.94	288.2	419	3 257
	MSRO	18.1	13.3	0.98	269.4	178	3 206
	Hybrid	10.1	5.9	0.72	123.8	32	2 333
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
	RMCD	0.29	0.26	0.03	2.5	29	75
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	1 401
	MSRO	8.7	3.8	0.28	57.6	41	1 005
	Hybrid	7.7	3.5	0.32	48.7	22	936
lhr14c	Original	24.5	23.9	0.80	235.7	276	2 719
	MSRO	19.0	9.2	0.58	149.3	67	2 198
	Hybrid	17.0	8.2	0.56	128.8	94	2 040
meg1	Original	27.9	27.7	0.50	579.5	2921	1 623
	MSRO	14.1	11.5	0.28	273.9	824	1 114
	Hybrid	12.0	10.5	0.33	246.2	189	1 226
	RMCD	2.4	2.3	0.17	16.6	202	575
onetone2	Original	50.6	49.5	2.65	683.8	238	7 572
	MSRO	46.4	40.8	2.46	826.8	209	6 755
	Hybrid	65.1	60.2	3.25	1757.4	145	9 834
orani678	Original	28.9	28.6	0.59	892.2	1605	2 284
	MSRO	28.2	11.2	0.26	125.0	2117	808
	Hybrid	21.9	12.4	0.37	298.0	747	1 255
	RMCD	4.1	3.9	0.22	84.2	368	771
rdist1	Original	4.7	4.6	0.27	90.3	30	1 050
	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 11. The results of reordering the rows for the frontal solver MA42 using the MSRO and the hybrid MSO algorithms. (CRAY J932)

		CPU Time (seconds)		
Identifier	Ordering	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
	RCMD	0.55	0.25	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	11.9	3.1	0.07
	RMCD	2.2	1.6	0.05
onetone2	Original	21.3	18.9	0.67
	MSRO	43.4	17.9	0.62
	Hybrid	44.6	23.5	0.75
orani678	Original	8.5	8.1	0.13
	MSRO	127.1	3.4	0.06
	Hybrid	70.9	3.3	0.08
	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

pseudo-diameter that provides start and end nodes for the MSRO algorithm. For this reason, if the trade-off 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. This is particularly true if a series of matrices with the same sparsity pattern are to be factorized since the row ordering needs only to be performed once.

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 9. 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

1. S. Barnard, A. Pothen and H. Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numer. Lin. Alg. Appl.*, 2, 317–198, 1995.
2. K. Camarda. *Ordering strategies for sparse matrices in chemical process simulation*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.
3. K. Cliffe, I. Duff and J. Scott. Performance issues for frontal schemes on a cache-based high performance computer. *Int. J. Numer. Methods in Engng.*, 42, 127–143, 1998.
4. A. Coon. *Orderings and direct methods for coarse granular parallel solutions in equation-based flowsheeting*. PhD thesis, University of Illinois, 1990.
5. A. Coon and M. Stadtherr. Generalized block-triangular matrix orderings for parallel computation in process flowsheeting. *Comput. Chem. Eng.*, 96, 787–805, 1995.
6. T. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, 97(23), 1997.
7. J. Dongarra, J. DuCroz, I. Duff and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1), 1–17, 1990.
8. I. Duff. MA32—a package for solving sparse unsymmetric systems using the frontal method. Report AERE R10079, Her Majesty's Stationery Office, London, 1981.
9. I. Duff. Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *SIAM J. Sci. Stat. Comput.*, 5, 270–280, 1984.
10. I. Duff, R. Grimes and J. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (release I). Technical Report RAL-TR-92-086, Rutherford Appleton Laboratory, 1992.
11. I. Duff and J. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Math. Soft.*, 22(1), 30–45, 1996.
12. I. Duff and J. 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.
13. N. Gibbs, W. Poole and P. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM J. Numer. Anal.*, 13, 236–250, 1976.
14. Harwell Subroutine Library. *A Catalogue of Subroutines (Release 12)*. Advanced Computing Department, AEA Technology, Harwell Laboratory, Oxfordshire, England, 1996.
15. P. Hood. Frontal solution program for unsymmetric matrices. *Int. J Numer. Methods in Engng.*, 10, 379–400, 1976.
16. B. Irons. A frontal solution program for finite-element analysis. *Int. J Numer. Methods in Engng.*, 2, 5–32, 1970.
17. G. Kumfert and A. Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT*, 18, 559–590, 1997.

18. J. Liu and A. Sherman. Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices. *SIAM J. Numer. Anal.*, 13, 198–213, 1976.
19. B. Mayoh. A graph technique for inverting certain matrices. *Math. Comput.*, 19, 644–646, 1965.
20. J. Reid and J. Scott. Ordering symmetric sparse matrices for small profile and wavefront. Technical Report RAL-TR-98-016, Rutherford Appleton Laboratory, 1998. Tp appear in *Int. J. Numer. Methods in Engng.*
21. J. Scott. Element resequencing for use with a multiple front algorithm. *Int. J. Numer. Methods in Engng.*, 39, 3999–4020, 1996.
22. J. Scott. Exploiting zeros in frontal solvers. Technical Report RAL-TR-98-041, Rutherford Appleton Laboratory, 1997.
23. J. Scott. A new row ordering strategy for frontal solvers. Technical Report RAL-TR-98-056, Rutherford Appleton Laboratory, 1998.
24. J. Scott. On ordering elements for a frontal solver. Technical Report RAL-TR-98-031, Rutherford Appleton Laboratory, 1998. To appear in *Comm. numer. methods eng.*
25. S. Sloan. An algorithm for profile and wavefront reduction of sparse matrices. *Int. J. Numer. Methods in Engng.*, 23, 1315–1324, 1986.
26. S. Sloan. A Fortran program for profile and wavefront reduction. *Int. J. Numer. Methods in Engng.*, 28, 2651–2679, 1989.
27. B. Hendrickson and R. Leland. The Chaco user's guide: version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1995.

