RAL-TR-1999-035

Row ordering for frontal solvers in chemical process engineering.

by

Jennifer A. Scott

Abstract

The solution of chemical process engineering problems often requires the repeated solution of large sparse linear systems of equations that have a highly asymmetric structure. The frontal method can be very efficient for solving such systems on modern computer architectures because, in the innermost loop of the computation, the method exploits dense linear algebra kernels, which are straightforward to vectorize and parallelize. However, unless the rows of the matrix can be ordered so that the frontsize is never very large, frontal methods can be uncompetitive with other sparse solution methods. We review a number of row ordering techniques that use a graph theoretical framework and, in particular, we show that a new class of methods that exploit the row graph of the matrix can be used to significantly reduce the frontsizes and greatly enhance frontal solver performance. Comparative results on large-scale chemical process engineering matrices are presented.

Keywords: chemical process simulation, ordering rows, frontal method, row graphs.

Computational Science and Engineering Department,
Atlas Centre, Rutherford Appleton Laboratory,
Oxon OX11 0QX, England.

May 11, 1999.

Contents

1 Introduction

In large-scale chemical process simulation the most computationally expensive step is generally the solution of large sparse systems of linear equations. The basic idea is to model the chemical process by a single very large nonlinear system of equations, with many thousands of variables. These equations are then solved using some variant of Newton's method. In the case of dynamic simulation, Newton's method (or variant) is applied at each time step. The use of Newton's method involves solving many systems of linear equations having the same sparsity structure. Solving the linear systems can represent over 80 per cent of the total computational cost on industrial problems (for example, Zitney, Brull, Lang and Zellet, 1995) and so it needs to be done as efficiently as possible. Any reduction in the linear system solution time will result in a significant saving in the total simulation time, allowing the solution of problems which would otherwise be intractable, as well as potentially enabling larger problems to be solved in a given time frame.

Unfortunately, process simulation matrices do not possess any desirable structural or numerical properties such as symmetry, positive definiteness, diagonal dominance, or bandedness, that are often associated with sparse matrices and are exploited in the development of efficient algorithms for the solution of sparse linear systems. The frontal method can be used to solve general sparse linear systems and work by Vegeais and Stadtherr (1990) and Zitney and Stadtherr (1993) has demonstrated the potential of the method for process simulation problems. A key feature of the frontal method is that, in the innermost loop of the computation, dense linear algebra kernels can be exploited. These are straightforward to vectorize and parallelize and are able to exploit high level BLAS kernels (Dongarra, DuCroz, Duff and Hammarling, 1990). This makes the method attractive for a wide range of modern computer architectures, including RISC based processors and shared memory parallel processors. However, the performance of the frontal method is highly dependent on the ordering of the rows of the matrix. The natural unit-block structure of process engineering problems can sometimes provide a reasonable ordering, and this has allowed the frontal method to be used with some success on these problems (Zitney and Stadtherr, 1993). But for many problems the natural ordering can be improved on. Furthermore, in most commercial software, unit-block structure information is not available to the linear system solver. Thus a general approach to reordering is needed. In this paper we review row ordering strategies and, in particular, show that the new class of methods introduced in a recent paper by Scott (1998) can yield substantial improvements in the performance of the frontal method for chemical process engineering problems.

This paper is organised as follows. In Section 2, we provide some background information on the frontal method and recall elementary concepts from graph theory that are useful in developing matrix ordering schemes. In Section 3, we review row ordering strategies that have been used in recent years for chemical processing problems. The new MSRO algorithms of Scott (1998) that exploit row graphs are described in Section 4. In Section 5, numerical results comparing the performance of the different approaches and their use with frontal solvers are presented. The design of a new library code `MC62` that implements the MSRO algorithm is discussed briefly in Section 6 and, finally, some concluding remarks are made in Section 7.

2 Background

2.1 The frontal method

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

$$Ax = b, \tag{2.1}$$

where the $n \times n$ matrix A is large and sparse. Although the method was originally developed in the 1970s 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). 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. The system (2.1) can be solved by a simple forward substitution

$$Ly = Pb,$$

followed by a backsubstitution

$$Uz = y.$$

The required solution

$$x = Qz$$

follows. At each stage of the matrix factorization, 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.

In general, the frontal matrix F is a rectangular matrix. The number of rows and columns in the frontal matrix (the *row* and *column frontsizes*) depends upon the number of rows of A that have been assembled and the number of eliminations that have been performed. 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

$$F = \begin{pmatrix} F_1 & F_2 \end{pmatrix}, \quad F_1 = \begin{pmatrix} F_{11} \\ F_{21} \end{pmatrix}, \quad F_2 = \begin{pmatrix} F_{12} \\ F_{22} \end{pmatrix}, \tag{2.2}$$

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_{21} and F_{12} are updated as

$$L_{21} = F_{21}U_{11}^{-1} \quad \text{and} \quad U_{12} = L_{11}^{-1}F_{12} \tag{2.3}$$

and then the Schur complement

$$F_{22} - L_{21}U_{12} \tag{2.4}$$

is formed. Finally, the factors L_{11} and U_{11}, as well as L_{12} and U_{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 et al., 1990) may be used. For example, the BLAS routine `GEMM` with interior dimension k can be used to form the Schur complement (2.4).

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 each 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

$$\max_{1 \le i \le n} frow_i \quad \text{and} \quad \max_{1 \le i \le n} fcol_i \tag{2.5}$$

  since these determine the amount of main memory needed,

- the mean row and column frontsizes

$$\frac{1}{n}\sum_{i=1}^{n} frow_i \quad \text{and} \quad \frac{1}{n}\sum_{i=1}^{n} fcol_i \tag{2.6}$$

  since these provide a measure of the factor storage

- the mean frontal matrix size

$$f_{avg} = \frac{1}{n}\sum_{i=1}^{n}(frow_i * fcol_i). \tag{2.7}$$

  A prediction of the number of floating-point operations that must be performed can be obtained from (2.7) (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 also define 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,

$$Life_i = \{\max_{1 \leq k,l \leq n} |l-k| : a_{ki} \neq 0 \text{ and } a_{li} \neq 0\}. \tag{2.8}$$

Camarda (1997) uses the sum of the lifetimes to compare the quality of different row ordering strategies: a small value for the sum of the lifetimes is used to indicate a good 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 paper, we shall be concerned only with running the frontal method on a single processor. Different ordering strategies should be considered when implementing a frontal algorithm in parallel. This is discussed, for example, by Camarda (1997), Mallya, Zitney, Choudhary and Stadtherr (1997), (1999), and Camarda and Stadtherr (1999), and, for element problems, by Scott (1996), and remains a subject for further investigation.

2.2 Graphs and matrices

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

A *graph* G is defined to be a pair $(V(G), E(G))$, where $V(G)$ is a finite set of *nodes* (or *vertices*) $v_1, v_2, ..., v_n$, and $E(G)$ is a finite set of *edges*, where an edge is a pair (v_i, v_j) of distinct nodes of $V(G)$. 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 $G = (V(G), E(G))$ with n nodes is a bijection of $\{1, 2, ..., n\}$ onto $V(G)$. The integer i $(1 \leq i \leq n)$ assigned to a node in $V(G)$ by a labelling is called the *label* (or *number*) of that node. Two nodes v_i and v_j in $V(G)$ are said to be *adjacent* (or *neighbours*) if $(v_i, v_j) \in E(G)$. The *degree* of a node $v_i \in V(G)$ is defined to be the number of nodes in $V(G)$ 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 $(v_{i_j}, v_{i_{j+1}}) \in E(G)$ 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 the following, we assume that the graphs we use are connected. If not, it is straightforward to apply the algorithms to each component of the graph.

We can 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.

3 Row ordering strategies

In recent years, a number of algorithms for automatically ordering matrices for frontal solvers have been proposed. In this section and the next, we briefly review these different strategies. Numerical results for the most promising approaches are included in Section 5.

3.1 Profile reduction algorithms

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 profile and bandwidth reduction 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). If the matrix A is numerically unsymmetric but has a symmetric sparsity pattern, an appropriate ordering of the rows for a frontal solver can be obtained using one of these profile reduction algorithms. The use of these algorithms can be extended to obtain orderings for unsymmetric matrices by applying them to the sparsity pattern of $A+A^T$. For matrices with an almost symmetric pattern, good orderings can generally be obtained using this approach (see, for example, Scott, 1998). But for highly asymmetric matrices, such as those that occur in process simulation, using the structure of $A+A^T$ does not yield useful results. This is because the number of entries in $A+A^T$ is almost twice that in A, indicating a large number of dependencies are introduced that do not exist in the actual problem.

3.2 P4 approach

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 highly asymmetric 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 so much as, for some examples, it can produce good orderings but for other problems, it can yield orderings that are significantly worse than the original ordering. Further results confirming this are given by Camarda and Stadtherr (1998). This inconsistency is possibly because the method places the rows with the largest number of entries early in the ordering which, in some cases, leads to a large column frontsize for many elimination steps. The P4 method was not, of course, developed with frontal solvers in mind. It is clear that, for frontal methods, specially developed algorithms are needed and, rather than a block triangular form, a variable band form is desirable.

3.3 RMCD ordering

The restricted minimum column degree (RMCD) ordering algorithm for reducing the size of the frontal matrix was recently discussed by Camarda (1997) and Camarda and Stadtherr (1998). This algorithm uses the concept of a net. A *net* n_l is defined to be a column l and all the rows i such that a_{il} is nonzero. This concept is useful because, when n_l 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. A simple example illustrating the RMCD algorithm is given in Camarda and Stadtherr (1998).

In his numerical experiments, Camarda (1997) 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 number of test examples from a variety of application areas (see also Scott, 1998 and Camarda and Stadtherr, 1998). Results for process engineering problems are included in Section 5.

3.4 RMNA ordering

The RMCD algorithm does not directly address the growth of the column frontsize. Experimental data reported by Scott (1998) shows that the reordered matrix can have a column frontsize that is many times that of the original matrix. To try and limit the column frontsize, Camarda (1997) proposes the restricted minimum net area (RMNA) algorithm. This algorithm is related to the RCMD algorithm but, rather than looking just at minimising the column degree when selecting the next net to be assembled, the RMNA algorithm is designed to restrict the additional area that will be added to the frontal matrix upon the assembly of a net. Specifically, at each stage, the RMNA algorithm chooses the column for which the product of the column degree and the net degree is a minimum, where the degree of the net n_l is defined to be the number of independent columns with nonzeros in the rows of n_l. Priority is given to the net whose degree was most recently updated.

The reported results of Camarda (1997) are disappointing. They show that the orderings obtained using the computationally more expensive RMNA algorithm generally offers little or no improvement on those obtained by the RCMD algorithm. It appears that the degree of n_l often provides a poor measure of the actual growth in the column frontsize that results from selecting n_l because of significant overlap between the columns with nonzeros in the rows of n_l and columns already in the front.

3.5 NMNC ordering

The RMCD and RMNA algorithms are local heuristic orderings: at each stage they choose the column that minimises a function based on the column and net degrees, 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) as the Minimum Net Cut (MNC) algorithm. The MNC algorithm is designed to order the matrix to bordered block diagonal form. It starts with a full transversal (zero-free diagonal) and employs row and column interchanges that maintain a full transversal. The bordered block diagonal form can be used to implement the frontal method in parallel (see Mallya et al., 1997). Recently, Camarda (1997) simplified the MNC algorithm. The so-called New Minimum Net Cut (NMNC) algorithm removes the full transversal restriction and, for the single processor frontal method, uses only row interchanges.

To describe the NMNC algorithm we need to recall the definition of the bipartite graph of a general square matrix of order n. 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 goal of the NMNC algorithm is to find a partitioning of the bipartite graph of A such that the number of nets cut by the partition is minimised, where the net n_l is said to be *cut* with respect to a partitioning of the rows of the matrix if column l has nonzero entries on both sides of the partition. The NMNC algorithm recursively partitions the rows of the matrix, so that the matrix is partitioned into 2, then into 4, and so on. For each partitioning, the rows are sorted according to their gain. The *gain* associated with moving a row j from one partition to another is defined to be the reduction in the net cut that results from the move. A negative gain indicates a move that increases the net cut. Rows that have been moved during the current partitioning are locked for the remainder of that partitioning. Two types of move are allowed: the first exchanges free (unlocked) rows between partitions, the second moves a single row into the other partition. Only moves with a positive gain are permitted. For each level of the partitioning, moves continue until no more rows can be moved.

The NMNC algorithm is more expensive to implement than the simple RMCD algorithm but the results presented in the thesis of Camarda (1997) show that it performs more consistently and can yield better orderings. This suggests that this method may be particularly useful when several factorizations follow the initial reordering.

4 Row graph ordering techniques

In the previous section, we considered both local and global reordering schemes. In this section, we look at a class of methods that use local ordering to refine a global ordering.

4.1 Row graphs

For developing row permutations of unsymmetric matrices, an alternative to using the digraph or the bipartite graph, is to use a row graph. Row graphs were first introduced by Mayoh (1965) and have recently been used by Scott (1998) for developing row orderings for frontal solvers.

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.

4.2 The MSRO algorithm

The MSRO row ordering algorithms introduced by Scott (1998) have their origins in the profile reduction algorithm of Sloan (1986) for symmetric matrices. The MSRO algorithms use the row graph and comprise two distinct phases:

- selection of a global ordering
- row reordering.

Selecting an appropriate global ordering is discussed below. The global ordering defines the global priority of each row. The row with the lowest global priority is chosen as the *start* row (that is, the row that is first in the global ordering is ordered first in the new ordering). In the second phase of the algorithm, the global ordering is used to guide the reordering. Rows with a high global priority will be chosen towards the end of the ordering.

A row is defined to be *active* if it has not yet been reordered but is adjacent in the row graph to a row that has already been reordered. The MSRO algorithm aims to reduce the row and column frontsizes by reducing the number of rows that are active at each stage and this is done by local reordering of the global ordering. For each row $i \in G_R$, the MSRO algorithm computes the priority function

$$P_i = W_1 * rcgain_i + W_2 * g_i. \tag{4.1}$$

Here W_1 and W_2 are positive integer weights, g_i is the (positive) global priority for row i, and $rcgain_i = rgain_i + cgain_i$, where $rgain_i$ and $cgain_i$ are the increases to the row and column frontsizes resulting from assembling (ordering) 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,

$$rgain_i = 1 - s_i \tag{4.2}$$

and

$$cgain_i = newc_i - s_i, \tag{4.3}$$

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

$$rcgain_i = 1 + newc_i - 2s_i \tag{4.4}$$

and this is minimised if row i brings a small number of new columns into the front and results in a large number of columns becoming fully summed.

The start row is ordered first then, at each stage, the next row in the ordering is chosen from a list of eligible rows to minimise P_i. The *eligible rows* are defined to be those that are active together with their neighbours. A list of eligible rows is maintained using the connectivity lists for the row graph. Thus, the MSRO algorithm attempts to keep a balance between having only a small number of rows and columns in the front and including rows that have a low global priority. The balance is determined by the choice of weights (see Section 4.4).

We note that the MSRO scheme has more freedom when choosing the next row to be assembled than the RCMD and RMNA algorithms. Once a column has been selected, the RCMD and RMNA algorithms assemble **all** the rows with nonzeros in that column, so that a block of rows rather than a single row is chosen at once. The MSRO approach selects one row and then, when choosing the next row, takes into account the effect of the previous choices.

4.3 The global ordering

The success of the MSRO algorithm is dependent upon first computing an appropriate global ordering. We consider three possible choices: the pseudodiameter, the spectral ordering, and the NMNC ordering.

4.3.1 The pseudodiameter

The *distance* between nodes i and j in an undirected graph 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,

$$D(G) = \max\{d(i,j) : i,j \in V(G)\}. \tag{4.5}$$

A *pseudodiameter* $\delta(G)$ is defined by any pair of nodes i and j in $V(G)$ for which $d(i,j)$ is close to $D(G)$. Experience has shown that the ends of a pseudodiameter provide good candidates for the starting nodes for profile and wavefront reduction algorithms and for bandwidth reduction algorithms (see, for example, Gibbs, 1976, Gibbs, Poole and Stockmeyer, 1976, and Sloan, 1986).

A pseudodiameter may be found using level set structures. A *level structure rooted at a node* r is defined as the partitioning of $V(G)$ 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 $l_1(r), l_2(r), ..., l_{i-1}(r)$.

The procedure that we use to locate a pseudodiameter is a modification of that described by Gibbs et al. (1976). Full details are given in Reid and Scott (1998).

Cuthill and McKee (1969) proposed that the ordering associated with the level-set structure be used as a basis for ordering for the variable-band method. In an earlier paper (Scott, 1998), we looked at applying the Reverse Cuthill-McKee algorithm to the row graph G_R of A. However, we found that improved orderings could be obtained by using the pseudodiameter of G_R as the global ordering within the MSRO algorithm. One end s of the pseudodiameter is chosen as the start row and is ordered first. The remaining rows are numbered according to their distance $d(i, s)$ from s, with those nearest to s being numbered first and row e numbered last. That is, g_i is chosen to be the distance $d(i, s)$.

4.3.2 Example

To illustrate the MSRO method with the pseudodiameter global ordering we use the matrix with the sparsity pattern given in Figure 4.1. We will use weights $(W_1, W_2) = (2, 1)$. For this matrix, the lifetimes are 3, 3, 3, 5, 4, 4 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 is 15 for this example.

Figure 4.1: The original matrix.

$$
\begin{array}{ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & x & & x & x & & \\
2 & & x & & x & x & \\
3 & x & & x & x & & x \\
4 & & x & & & & \\
5 & & & & x & x & x \\
6 & & & & & & x
\end{array}
$$

The start and target end rows (s, e) are chosen to be $(4, 6)$ since, by inspection, $d(4, 6) = 3$ and $d(i, j) \leq 3$, $i, j = 1, 2, ..., 6$). The initial priorities are given in Table 4.1. Note that initially $rcgain_i$ is just one more than the number of entries in row i.

Table 4.1: Initial priorities for MSRO method.

Row	$rcgain_i$	$d(i,4)$	P_i
1	4	2	10
2	4	1	9
3	5	2	12
4	2	0	4
5	4	2	10
6	2	3	7

Row 4 is ordered first. Only row 2 is a neighbour of row 4 so only its priority changes. Its priority decreases by W_1 to 7. We now have two rows with a priority value of 7 but only row 2 is active so it is ordered next. Row 2 brings columns 4 and 5 into the front and rows 1, 3, and 5 become active. Since row 1 has an entry in column 4, its priority decreases by W_1. The priority of row 3 is also decreased by W_1 and, because row 5 has entries in both columns 4 and 5, its priority decreases by $2 * W_1$, resulting in the matrix of Figure 4.2 (rows with the priority given as $-$ have been reordered).

Figure 4.2: Partially ordered matrix.

$$\begin{array}{ccccccc c}
 & 1 & 2 & 3 & 4 & 5 & 6 & Priority \\
4 & & x & & & & & - \\
2 & & x & & x & x & & - \\
1 & x & & x & x & & & 8 \\
3 & x & & x & x & & x & 10 \\
5 & & & & x & x & x & 6 \\
6 & & & & & & x & 7
\end{array}$$

Figure 4.3: Partially ordered matrix.

$$\begin{array}{ccccccc c}
 & 1 & 2 & 3 & 4 & 5 & 6 & Priority \\
4 & & x & & & & & - \\
2 & & x & & x & x & & - \\
5 & & & & x & x & x & - \\
1 & x & & x & x & & & 8 \\
3 & x & & x & x & & x & 8 \\
6 & & & & & & x & 5
\end{array}$$

Row 5 now has the lowest priority value and so is ordered next, bringing column 6 into the front and making row 6 active. The priorities of rows 3 and 6, which have entries in column 6, are decreased by W_1, giving the matrix in Figure 4.3.

Figure 4.4: Final reordered matrix.

$$
\begin{array}{ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
4 & & x & & & & \\
2 & & x & & x & x & \\
5 & & & & x & x & x \\
6 & & & & & & x \\
3 & x & & x & x & & x \\
1 & x & & x & x & &
\end{array}
$$

We next order row 6 because it has the lowest priority value. The priority of row 3 then decreases so that it is ordered ahead of row 1. The final reordered matrix is given in Figure 4.4. The sum of the lifetimes for the reordered matrix is 16.

4.3.3 Spectral ordering

Spectral algorithms have been used in recent years for profile and wavefront reduction of symmetrically structured matrices. 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,

$$
L = \{l_{ij}\} = \begin{cases} -1 & \text{if } i \neq j \text{ and } \quad s_{ij} \neq 0 \\ 0 & \text{if } i \neq j \text{ and } \quad s_{ij} = 0 \\ \sum_{i \neq j} |l_{ij}| & \text{if } i = j. \end{cases} \tag{4.6}
$$

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.

Experience has shown that spectral orderings often do well in a global sense but can perform poor locally (see Kumfert and Pothen, 1997). We therefore use the spectral ordering to provide a global ordering for the priority function (4.1). Specifically, we choose the start row to be the first row in the spectral ordering and, for a matrix with n rows, we take the second term in the priority function (4.1) to be

$$
g_i = (h/n)p_i. \tag{4.7}
$$

Here p_i is the position of row i in the spectral ordering and h is the number of level-sets in the level set structure rooted at the start row. The normalization of p_i results in g_i varying up to h, which is at most the length of the pseudodiameter. Without normalization, the second term in the priority function would have a much larger influence than it does when the pseudodiameter is used as the global ordering.

4.3.4 NMNC ordering

The NMNC ordering, which was discussed in Section 3.5, can also be used as the global ordering for the second phase of the MSRO algorithm. As for the spectral ordering, we take the second term in the priority function to be (4.7), where p_i is now the position of row i in the NMNC ordering and h is again the depth of the level set structure rooted at the start row.

4.4 Choice of weights

The performance of the MSRO algorithms is dependent on the choice of the integer weights (W_1, W_2). The detailed numerical experiments performed by Scott (1998) show that no single choice of weights is best for all problems. Based on numerical results for a wide range of practical problems, when the pseudodiameter is used as the global ordering for the MSRO algorithm, Scott recommends trying the weights $(2,1)$ and $(32,1)$ and selecting the better result. Using the spectral ordering, Scott proposes the weights $(1,2)$ and $(32,1)$, unless the matrix has a short pseudodiameter. In this case, the best results are achieved with a larger value of W_2, so that the ordering more closely follows the spectral ordering. When the NMNC ordering is used as the global ordering we also use the weights $(1,2)$ and $(32,1)$ and take the better result.

4.5 Reversing the row order

If we assume that for a given row ordering the rows have been relabelled $1, 2, ..., n$, then the reverse ordering assembles the rows in the order $n, ..., 2, 1$. It can be shown that the sum of the lifetimes is independent of whether the rows are assembled in the given order or in the reverse order. Moreover, Reid and Scott (1999) prove that the maximum and mean column frontsizes are invariant if the row order is reversed. However, the maximum and mean row frontsizes and the mean frontal matrix size f_{avg} are, in general, different for the reverse order. Numerical experimentation has shown that, for some examples, f_{avg} can be significantly reduced by reversing a given row order while for other examples, the converse is true. We therefore compute the mean frontal matrix size for the MSRO orderings and also for the reverse MSRO orderings and select the ordering for which f_{avg} is the smaller.

5 Numerical results

In this section, we first describe the chemical engineering problems that we use for testing the row ordering algorithms discussed in this paper and then present numerical results.

5.1 Test problems

The test problems are listed in Table 5.1. Each problem comes from chemical process engineering. Problems marked with a † are from the University of Florida Sparse Matrix Collection (Davis, 1997, see http://www.cise.ufl.edu/~davis/sparse/) and those marked by ‡ are from the Harwell-Boeing Collection (Duff, Grimes and Lewis, 1992, see http://www.dci.clrc.ac.uk/Activity/SparseMatrices). The remaining problems were supplied by Mark Stadtherr of the University of Notre Dame; further details of these problems may be found in Mallya et al. (1997). In addition to the order of the matrix and the number of entries in the matrix, we give the symmetry index and information on the row graph of the matrix (the length of

Table 5.1: The test problems. † indicates problem taken from University of Florida Sparse Matrix Collection and ‡ indicates from Harwell-Boeing Collection.

Identifier	Order	Number of entries	Symmetry index	Pseudo diameter	Edges in row graph ($*10^3$)	Average number neighbours
`4cols`	11770	43668	0.0159	64	210	17.8
`10cols`	29496	109588	0.0167	156	527	17.9
`bayer01`†	57735	277774	0.0002	154	1532	26.5
`bayer03`†	6747	56196	0.0031	42	400	59.3
`bayer04`†	20545	159082	0.0016	44	1099	53.5
`bayer09`†	3083	21216	0.0212	30	142	46.0
`ethylene-1`	10673	80904	0.2973	21	2036	190.7
`ethylene-2`	10353	78004	0.3020	21	1832	176.9
`extr1`†	2837	11407	0.0042	57	37	13.1
`hydr1`†	5308	23752	0.0041	54	96	24.2
`icomp`	69174	301465	0.0010	301	1833	18.1
`lhr07c`†	7337	156508	0.0174	49	704	96.0
`lhr14c`†	14270	307858	0.0066	41	1394	97.7
`lhr17c`†	17576	381975	0.0015	41	1731	98.5
`lhr34c`†	35152	764014	0.0015	49	3464	98.5
`lhr71c`†	70304	1528092	0.0016	72	6930	98.6
`meg1`†	2904	58142	0.0024	7	372	128.1
`radfr1`†	1048	13299	0.0537	29	395	37.7
`rdist1`†	4134	94408	0.0588	54	322	78.0
`rdist2`†	3198	56934	0.0456	54	188	58.7
`rdist3a`†	2398	61896	0.1404	29	216	90.0
`west2021`‡	2021	7353	0.0033	15	38	18.7

the pseudodiameter of the row graph, the number of edges in the graph, and the average number of neighbours each row has). The *symmetry index* of a matrix A is defined as

$$s(A) = 1 - \frac{nz(A + A^T - D) - nz(A - D)}{nz(A - D)},$$

where $nz(A - D)$ and $nz(A + A^T - D)$ denote the number of off-diagonal entries in A and $A + A^T$, respectively. Thus $s(A)$ is measure of how far from symmetry the sparsity pattern of A is. Small values indicate a high degree of asymmetry. We see that all the chosen test problems are highly asymmetric. The pseudodiameter was computed using the `MC62` code (see Section 6).

The reported results of Camarda (1997) and Camarda and Stadtherr (1998) suggest that, of the row ordering algorithms discussed in Section 3, the most promising approaches for unsymmetric problems are the RMCD and NMNC algorithms. We therefore restrict our numerical experiments to the RMCD, NMNC, and MSRO algorithms. In the following, MSRO+pseudodiameter denotes the MSRO algorithm with the pseudodiameter used as the global ordering. MSRO+spectral and MSRO+NMNC are defined similarly. For the NMNC algorithm, we use the code of Camarda (1997), for which a complete listing is given in his thesis. For the other algorithms, we use a new code that will be included in the next release of the Harwell Subroutine Library (http://www.dci.clrc.ac.uk/Activity/HSL). The new code is written in standard Fortran 77 and is called `MC62`. We briefly discuss the design of `MC62` in Section 6. In our experiments involving the spectral method, the Fiedler vector of the row graph was obtained using Version 2.0 of the Chaco package (Hendrickson and Leland, 1995).

Unless indicated otherwise, the numerical results were obtained using the EPC (Edinburgh Portable Compilers, Ltd) Fortran 90 compiler with optimization -O running on a 143 MHz Sun Ultra 1.

5.2 A comparison of the methods

In this section, we compare the performance of the different row ordering algorithms. In Table 5.2, the mean frontal matrix size is given. For comparison, we include the mean frontal matrix size for the original ordering. In Table 5.3, we present the sum of the lifetimes as a percentage of the sum of the lifetimes of the original ordering. We highlight in bold the smallest values for each problem.

For 4 of the largest problems, `10cols`, `bayer01`, `icomp`, and `lhr71c`, we were unable to obtain a spectral ordering with the Chaco package and for these problems no results for MSRO+spectral are available.

The main conclusion that we can draw from our results is that, when a spectral ordering is available, the best results are generally achieved using the MSRO+spectral algorithm. We now examine our findings in a little more detail. We first note that the performance of the RMCD algorithm can vary greatly between problems. In general, all the other algorithms perform better than RMCD: there is only one problem, `megl`, for which RMCD gives the smallest value of f_{avg}. Moreover, for a large proportion of the test problems, the RMCD

Table 5.2: The mean frontal matrix size ($f_{avg} * 10^2$) for the different reordering algorithms. The smallest values are highlighted. † denotes spectral ordering not available.

Identifier	Original	RMCD	NMNC	MSRO+ pseudo diameter	MSRO+ spectral	MSRO+ NMNC
`4cols`	2218	361	982	**30**	45	61
`10cols`	7091	448	2422	**39**	†	87
`bayer01`	1183	27136	992	182	†	812
`bayer03`	200	438	195	27	**12**	234
`bayer04`	1911	4162	1683	334	**59**	972
`bayer09`	249	152	230	20	**12**	178
`ethylene-1`	1452	11249	573	3910	2449	**213**
`ethylene-2`	451	10496	292	2818	569	**67**
`extr1`	49	486	34	4	**3**	18
`hydr1`	310	231	197	10	**3**	58
`icomp`	1217	1274	847	73	†	198
`lhr07c`	521	2180	150	62	**48**	130
`lhr14c`	1076	7645	266	153	**134**	224
`lhr17c`	1329	11506	275	200	**170**	255
`lhr34c`	1499	48940	1499	283	**172**	472
`lhr71c`	1548	204070	1548	835	†	486
`meg1`	11823	**461**	3068	1837	1015	1781
`radfr1`	36	**4**	5	**4**	**4**	5
`rdist1`	146	1251	20	**17**	20	**17**
`rdist2`	65	13347	11	13	**10**	**10**
`rdist3a`	91	8262	**22**	36	**22**	30
`west2021`	179	28	151	**4**	**4**	40

algorithm produces orderings for which the sum of the lifetimes is actually greater than for the original ordering.

Although more consistent, for many problems the NMNC algorithm is only able to achieve relatively modest reductions in the size of the frontal matrix. However, the NMNC orderings are improved significantly when used in conjunction with the MSRO algorithm. Comparing the columns headed 'NMNC' and 'MSRO+NMNC' in Tables 5.2 and 5.3, we see that for most problems the MSRO+NMNC algorithm outperforms the NMNC algorithm and, for some problems, including `4cols` and `10cols`, the improvements are dramatic.

Comparing the use of the different global orderings with the MSRO algorithm we see that for most, but not all, of the problems the pseudodiameter gives better results than using the NMNC ordering, while in turn the spectral ordering is better than the pseudodiameter. There are only two problems, `ethylene-1` and `ethylene-2`, for which the MSRO algorithm with the pseudodiameter and the spectral ordering perform poorly. To try and gain some insight into why this is, we need to look at the row graphs for these matrices. We see from Table 5.1 that for these problems the average number of neighbours each row has is large and, compared with the order of the matrices, the pseudodiameter is short. It would appear that the MSRO algorithm used with the pseudodiameter or spectral ordering does well provided the rows have only a small number of neighbours; where there is a high degree of connectivity

Table 5.3: The sum of the lifetimes for the different reordering algorithms as a percentage of the sum of the lifetimes of the original ordering. The smallest values are highlighted. $*$ indicates the sum of the lifetimes is more than one thousand times greater than for the original ordering. † denotes spectral ordering not available.

Identifier	RMCD	NMNC	MSRO+ pseudo diameter	MSRO+ spectral	MSRO+ NMNC
`4cols`	71	70	**13**	15	18
`10cols`	45	61	**8**	†	12
`bayer01`	$*$	92	35	†	80
`bayer03`	609	98	36	**24**	104
`bayer04`	517	94	45	**18**	67
`bayer09`	221	94	30	**24**	82
`ethylene-1`	631	60	187	131	**34**
`ethylene-2`	$*$	71	297	119	**44**
`extr1`	498	82	28	**27**	67
`hydr1`	200	81	17	**11**	35
`icomp`	$*$	79	**25**	†	42
`lhr07c`	941	65	47	**44**	65
`lhr14c`	$*$	57	55	**49**	60
`lhr17c`	$*$	54	57	**47**	57
`lhr34c`	$*$	100	61	**44**	69
`lhr71c`	$*$	100	89	†	69
`meg1`	62	63	43	**31**	42
`radfr1`	33	35	**30**	**30**	30
`rdist1`	170	33	**29**	30	**29**
`rdist2`	947	31	31	**27**	28
`rdist3a`	806	51	61	**45**	51
`west2021`	96	96	**14**	**14**	49

between the rows one of the other algorithms may perform better and we do not recommend using the pseudodiameter or spectral ordering.

A comparison of the results in Tables 5.2 and 5.3 shows that if the best ordering is selected on the basis of the sum of the lifetimes then for a number of problems a different ordering is chosen than would be chosen if the mean frontal matrix size was used. For example, for problem `meg1` the mean frontal matrix size for the RCMD ordering is significantly smaller than for all the other algorithms but if the sum of the lifetimes was to be used, the MSRO+spectral ordering would be chosen. Similarly, for `bayer09` the sum of the lifetimes is smaller for NMNC than for RCMD but f_{avg} for RCMD is smaller than for NMNC. Again, for `west2021`, for RCMD and NMNC the sum of the lifetimes is 96 per cent of the original but the RCMD has a much smaller mean frontal matrix size. Although the sum of the lifetimes has been used in the past as the measure for selecting a good ordering (Camarda, 1997), on the basis of our findings and the results of Reid and Scott (1999), we recommend using the mean frontal matrix size.

5.3 Use with frontal solvers

As discussed in the introduction, the main motivation behind this work is the need for row orderings to improve the efficiency of frontal solvers. We now present results that illustrate the effect on frontal solver factorization times of preordering the rows.

5.3.1 `MA42`

In the Harwell Subroutine Library (1996), the `MA42` package (Duff and Scott, 1996) is a general purpose frontal solver. 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 here. In Table 5.4 we present the CPU time (in seconds) taken by `MA42` to factorize the reordered matrices. The timings include the i/o overhead for using direct access files to hold the matrix factors, but do not include the time required to reorder the rows. Partial pivoting and Level 3 BLAS are used by `MA42`. In our experiments, we use a minimum pivot block size of 16 together with a version of `MA42` that attempts to exploit blocks of zeros within the front (see Scott, 1997 for details). Once the factors have been computed, a separate subroutine is used to perform the forward and back substitutions needed to complete the solution. Thus any number of systems with the same factors but different right-hand sides can be solved for, either simultaneously or one at a time. Timings are not included where the results of the previous section have shown an ordering is not as good as the original ordering.

As expected, the results demonstrate that improved orderings generally lead to savings in the factorization time. For most problems, we have been able to achieve savings of more than 50 per cent compared with the original ordering and for some problems the factorization time has been reduced by as much as 80 per cent. We note however that the savings are not always as large as the reductions in the frontsize and in the lifetimes might lead us to expect. This is partly because `MA42` is able to offset some of the effects of a poor ordering by exploiting zeros within the frontal matrix (see also Duff and Scott, 1997 and Cliffe, Duff and Scott, 1998). Furthermore, the effect of using Level 3 BLAS means that the poorer orderings can achieve a higher Megaflop rate.

We have also compared the performance of `MA42` with that of the Harwell Subroutine Library sparse solver `MA48` (Duff and Reid, 1993, 1996). `MA48` is a general purpose Fortran 77 sparse code that uses Gaussian elimination for solving unsymmetric systems whose coefficient matrix need not be square. The analyse phase first permutes the matrix to block triangular form and then, on each submatrix of the block diagonal, uses a Markowitz criterion for maintaining sparsity and threshold partial pivoting for numerical stability. A subsequent factorize phase must then be used to generate the factors. There is a second factorize option ('fast' factorize) to rapidly factorize a matrix with the same sparsity structure as one previously factorized by the routine. The solve phase uses the computed factors to solve for a single right-hand side at a time. The factors are held in-core.

Table 5.4: The factorization times (in seconds) for `MA42` used with the different reordering algorithms (Sun Ultra). The fastest times are highlighted. $*$ indicates `MA42` not run because original ordering is better than reordering. † denotes spectral ordering not available.

Identifier	Original	RMCD	NMNC	MSRO+ pseudo diameter	MSRO+ spectral	MSRO+ NMNC
`4cols`	17.8	17.0	14.3	**2.8**	3.3	3.2
`10cols`	84.4	48.8	64.0	**7.6**	†	10.0
`bayer01`	43.2	$*$	40.5	**20.2**	†	119
`bayer03`	2.5	$*$	2.3	1.4	**1.1**	$*$
`bayer04`	21.8	$*$	20.0	10.6	**5.0**	12.6
`bayer09`	0.9	1.3	0.9	0.5	**0.4**	0.9
`ethylene-1`	7.5	$*$	7.9	$*$	$*$	**5.1**
`ethylene-2`	7.0	$*$	7.7	$*$	$*$	**3.0**
`extr1`	0.5	$*$	0.4	**0.3**	0.4	0.5
`hydr1`	1.7	2.0	1.4	0.8	**0.7**	1.7
`icomp`	41.4	$*$	33.2	**11.2**	†	15.7
`lhr07c`	11.8	$*$	7.4	3.8	**3.5**	7.1
`lhr14c`	23.9	$*$	13.1	9.2	**8.1**	12.6
`lhr17c`	23.4	$*$	23.3	**13.2**	**13.2**	16.6
`lhr34c`	214	$*$	$*$	218	**158**	237
`lhr71c`	**345**	$*$	$*$	487	†	399
`meg1`	27.2	**2.4**	34.6	20.8	14.7	21.2
`radfr1`	0.4	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
`rdist1`	4.7	$*$	1.4	**1.3**	**1.3**	**1.3**
`rdist2`	2.4	$*$	**0.8**	**0.8**	**0.8**	0.9
`rdist3a`	2.2	$*$	**0.9**	1.1	**0.9**	1.1
`west2021`	0.5	0.4	0.5	**0.2**	**0.2**	0.3

In Table 5.5, we present the time to reorder the matrix, the `MA42` factor and solve times, and for `MA48`, the analyse, factor, and fast factor times. There is no need to reorder the matrix prior to the `MA48` analyse phase thus the reorder time is only important when using `MA42`. The solve times are for a single right-hand side. Here (and in Table 5.6), for the test problems for which a spectral ordering is available and gives the smallest frontsizes, the reorder times are for MSRO+spectral (but the time taken to obtain the spectral ordering is not included as we do not currently have available a Fortran code to do this). For `ethylene-1` and `ethylene-2` the times are for MSRO+NMNC, and for `meg1` the time is for RCMD. For `lhr71c` the original ordering is retained.

We see that for `MA48`, the analyse phase (which must be performed once for each test problem) is more expensive than the factor phase. For a number of problems (including `4cols`, `10cols`, and the `lhr` problems), this leads to the `MA48` analyse+factor time being slower than reordering the matrix and factorising using `MA42`. As mentioned in the Introduction, for many chemical process engineering problems, a large number of factorizations of matrices having the same sparsity pattern is required. In this case, the significant times are the `MA42` factor time and the `MA48` factor and fast factor times. The `MA48` fast factorization uses the pivot sequence from a previous factorization and this may become unstable if the matrix entries are markedly different from the earlier call. Thus even if the matrix pattern remains unchanged, it may be necessary to

Table 5.5: Times (in seconds) for `MA42` and `MA48`.

Identifier	`MA42`			`MA48`			
	Reorder time	Factor	Solve	Analyse	Factor	Fast factor	Solve
`4cols`	0.6	2.8	0.35	5.5	2.1	1.6	0.08
`10cols`	2.1	7.6	0.91	15.0	4.7	3.2	0.20
`bayer01`	4.9	20.2	2.62	13.4	4.3	2.5	0.31
`bayer03`	1.3	1.1	0.18	1.2	0.5	0.3	0.04
`bayer04`	3.7	5.0	0.62	5.8	2.0	1.3	0.13
`bayer09`	0.5	0.4	0.07	0.3	0.2	0.1	0.02
`ethylene-1`	6.4	5.1	0.36	1.6	0.7	0.4	0.07
`ethylene-2`	5.8	3.0	0.24	1.6	0.7	0.4	0.07
`extr1`	0.1	0.3	0.05	0.2	0.07	0.03	0.01
`hydr1`	0.3	0.7	0.09	0.6	0.2	0.1	0.02
`icomp`	6.0	11.2	1.51	1.8	0.5	0.3	0.23
`lhr07c`	4.4	3.5	0.28	7.1	4.7	4.0	0.11
`lhr14c`	8.6	8.1	0.58	15.9	8.4	7.0	0.22
`lhr17c`	10.9	13.2	0.83	20.3	12.0	10.1	0.29
`lhr34c`	21.6	158	2.09	148	231	225	0.77
`lhr71c`	0.0	345	6.10	219	354	347	1.42
`meg1`	0.1	2.4	0.25	1.3	0.5	0.4	0.01
`radfr1`	0.1	0.2	0.02	0.4	0.2	0.1	0.01
`rdist1`	1.2	1.3	0.11	5.7	2.2	1.7	0.08
`rdist2`	0.6	0.8	0.06	2.3	0.9	0.6	0.04
`rdist3a`	0.9	0.9	0.08	1.9	0.8	0.6	0.04
`west2021`	0.1	0.2	0.03	0.1	0.03	0.01	0.01

generate a new pivot sequence and it is then the factor time that is important. For a few of problems, including `lhr07` and `lhr34c`, the `MA42` factor time is less than both the `MA48` factor and fast factor times. For a number of other problems, including `radfr1` and the `rdist` problems, the `MA42` factor times are competitive with the `MA48` factor times. But for most of our test problems, the `MA48` fast factor times are smaller than the `MA42` factor times. In addition, the solve times for `MA48` are significantly less than for `MA42`. However, when solving for a number of right-hand sides at once, `MA42` uses Level 3 BLAS so that the time for solving simultaneously for k right-hand sides can be much less than for k separate solves. For example, for problem `4cols` the time for solving for a single right-hand side is 0.35 seconds and for 10 right-hand sides it is 1.22 seconds. For `lhr34c` the corresponding times are 2.1 and 9.0 seconds. For `MA48` the time for k right-hand sides is k times the single solve time.

5.3.2 FAMP

We have also performed tests with the frontal solver FAMP. This solver was developed at the University of Illinois and at Cray Research, Inc. and described by Zitney and Stadtherr (1993) and Zitney et al. (1995). Unlike `MA42`, FAMP was specifically designed for assembled matrices (non-element form). Moreover, while `MA42` is written in standard Fortran 77 and is fully portable, FAMP has been finely tuned for Cray systems, including the use of assembly language kernels. As a result, on Cray machines, FAMP is faster than `MA42`. We compare the performance of FAMP on a single processor of a Cray J932 with that of

`MA48`. All timings are CPU times in seconds.

Table 5.6: Times (in seconds) for FAMP and `MA48` (Cray J932).

Identifier	FAMP				`MA48`			
	Reorder time	Original order	New order	Solve	Analyse	Factor	Fast factor	Solve
`4cols`	4.96	1.97	0.16	0.09	7.84	2.96	1.37	0.06
`10cols`	11.3	7.68	2.41	0.42	22.2	9.03	3.47	0.16
`bayer01`	27.9	10.2	6.58	0.82	36.4	10.2	3.66	0.32
`bayer03`	7.21	0.89	0.64	0.09	3.54	1.25	0.44	0.04
`bayer04`	20.2	4.37	2.18	0.27	15.9	4.33	1.59	0.11
`bayer09`	2.59	0.35	0.27	0.04	1.25	0.30	0.14	0.01
`ethylene-1`	22.4	1.76	1.71	0.16	5.24	1.80	0.57	0.06
`ethylene-2`	20.2	1.76	1.11	0.15	5.07	1.87	0.59	0.06
`extr1`	0.76	0.22	0.21	0.04	0.68	0.27	0.08	0.01
`hydr1`	2.00	0.56	0.41	0.07	1.87	0.76	0.26	0.03
`lhr07c`	22.8	2.20	1.48	0.12	15.8	5.65	2.41	0.06
`lhr14c`	44.9	4.71	3.21	0.22	34.1	12.1	4.85	0.11
`lhr17c`	56.4	5.85	4.53	0.30	41.8	14.2	5.74	0.14
`lhr34c`	113.0	12.2	8.80	0.59	97.6	34.8	16.1	0.29
`meg1`	0.61	3.41	0.75	0.04	2.69	1.19	0.50	0.06
`radfr1`	1.10	0.19	0.11	0.01	0.95	0.41	0.15	0.01
`rdist1`	12.5	1.23	0.80	0.07	11.1	4.17	1.74	0.03
`rdist2`	6.22	0.72	0.47	0.05	5.94	2.42	0.99	0.02
`rdist3a`	9.11	0.66	0.51	0.04	4.07	1.72	0.62	0.01
`west2021`	0.43	0.17	0.12	0.03	0.34	0.12	0.03	0.01

In Table 5.6, we present the time to reorder the matrix, the FAMP factor times for the original and new orderings, and solve time for the new ordering. For `MA48`, we present the analyse, factor, fast factor, and solve times (single right-hand side). Again, the reorder time is only significant for the frontal solver. For FAMP, the fast factor time is only slightly less than the factor time (see Zitney, Mallya, Davis and Stadtherr, 1996), so we do not quote this. We again see that reordering can substantially reduce the factor times for the frontal solver and this emphasizes the importance of obtaining good row orderings. However, we also observe that reordering the rows is more expensive on the Cray than factorizing the matrix and, if only a single matrix factorization is needed, it is faster to use FAMP with the original matrix ordering. Alternatively, since the reordering is quite separate from the frontal code, we can generate the ordering on another machine such as the SUN that has faster integer arithmetic and then pass the ordering to FAMP on the Cray. The analyse phase of `MA48` is again more expensive than the factor phase. For some problems, including `4cols`, `10cols`, and the `lhr` problems, the FAMP factor time for the new order is less than both the `MA48` factor and fast factor times. For the remaining problems, FAMP is faster than the `MA48` factorization but slower than `MA48` fast factorization. It should however be noted that, beyond using the vendor-supplied BLAS, `MA48` is not specifically tuned to run on the Cray. Again, solve times using `MA48` are faster than those for the frontal code, generally by a factor of 2 or 3.

We conclude that, with a good row ordering, frontal schemes can provide a powerful and competitive alternative to general-purpose sparse solvers for chemical process applications.

6 Design of `MC62`

In this section, we briefly discuss our new code `MC62` that implements the MSRO algorithm. The code will be included in the next release of the Harwell Subroutine Library (HSL 2000) and is available for use now under licence. Anyone interested in using the code may contact the author for details of terms and conditions.

The subroutines in the `MC62` package are named according to the naming convention of the Harwell Subroutine Library. Subroutine `MC62I` must first be called to provide default values for the parameters that control the execution of the package. If the user wishes to use values other than the defaults, the corresponding parameters should be reset after the call to `MC62I`. The main subroutine `MC62A` accepts the sparsity pattern of the matrix A, either in sparse row format or in sparse column format. `MC62A` performs full checks on the data and calls `MC62B` to compute statistics (the maximum and mean row and column frontsizes, the mean frontal matrix size, and the sum of the lifetimes) for the original row order. `MC62A` then either

- implements the MSRO algorithm, or
- implements the RMCD algorithm, or
- implements both the MSRO and RMCD algorithm and selects the better ordering.

The code offers both the MSRO and RCMD algorithms since, as we saw in our numerical experiments in Section 5 and in Scott (1998), RCMD can outperform if the row graph has a short pseudodiameter.

For the MSRO algorithm, subroutine `MC62C` is called to generate the row graph. This subroutine is also available as a separate entry. The user then has the option of either using the pseudodiameter of the row graph or specifying the global priority for each row. If used, the pseudodiameter is computed using routines from the Harwell Subroutine Library package `MC60`. The weights for the priority function (4.1) may be chosen by the user, otherwise default values based on the recommendations of Scott (1998) are used. If more than one set of weights is used or if both the MSRO and RMCD orderings are computed, the best row ordering is selected on the basis of the mean frontal matrix size.

Subroutine `MC62B` is also used to compute statistics for the rows taken in reverse order. If the mean frontal matrix size is smaller for the reverse order, the reverse order is returned to the user as the new row order. Note that `MC62B` is also available as a separate entry and by returning several statistics, the user can select the ordering on the basis of what he or she considers is most important for their application. For example, if minimising the amount of main memory needed by the frontal solver is the primary consideration, the user can compare orderings on the basis of the maximum frontsizes. If minimising factor storage is the main concern, the mean frontsizes should be used.

7 Conclusions

In this paper, we have looked at the problem of reordering the rows of a general unsymmetric matrix A for use with frontal solvers. We have reviewed recent algorithms and, in particular, have discussed variants of the MSRO algorithm. This approach is based on the row graph of A and uses a combination of a local and a global ordering scheme. We have found that the MSRO algorithm using the pseudodiameter or spectral ordering works well on a wide range of problems from chemical processing applications and, in general, produces orderings that are a substantial improvement on the original ordering and on the orderings obtained by the RMCD and NMNC algorithms. The only problems we have found that it does not work well on are those for which the row graph has a high degree of connectivity which leads, in turn, to a short pseudodiameter.

The results presented for the frontal solvers `MA42` and FAMP demonstrate that a good row ordering can lead to substantial reductions in the time taken to factorize a matrix. Of course, reordering the rows takes time and can dominate the overall solution time if a single factorization of the reordered matrix is performed (see also Scott, 1998). However, large-scale simulation or optimization models will typically be used many times. This is particularly true in an on-line operations environment. Even if the need for a matrix refactorization is relatively infrequent, over the lifetime of a process model the total number of factorizations of matrices with the same structure but different numerical values will still be large. In this case, the cost of a single matrix reordering represents an insignificant part of the total cost and investing in obtaining improved orderings is well worthwhile.

8 Acknowledgements

I would like to thank Kyle Camarda for supplying a copy of his code that implements the NMNC algorithm. This code was used to obtain the NMNC results included in Section 5. I am also grateful to Mark Stadtherr of the University of Notre Dame for providing me with some of the test problems and for helpful comments on a draft of this paper.

References

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 Illinois at Urbana-Champaign, 1997.

K.V. Camarda and M.A. Stadtherr. Frontal solvers for process engineering: local row ordering strategies. *Computers and Chemical Engineering*, **22**, 333–341, 1998.

K.V. Camarda and M.A. Stadtherr. Matrix ordering strategies for process engineering: graoh partitioning algorithms for parallel computation. *Computers and Chemical Engineering*, 1999. to appear.

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.

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. Technical Report RAL-TR-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. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Mathematical Software*, **22**(1), 30–45, 1996.

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

I.S. Duff, 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. A hybird profile reduction algorithm. *ACM Trans. Mathematical Software*, **2**, 378–387, 1976.

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.U. Mallya, S.E. Zitney, S. Choudhary, and M. A. Stadtherr. A parallel block frontal solver for large scale process simulation: reordering effects. *Computers and Chemical Engineering*, **21**, S439–S444, 1997.

J.U. Mallya, S.E. Zitney, S. Choudhary, and M. A. Stadtherr. Matrix reordering effects on a parallel frontal solver for large scale process simulation. *Computers and Chemical Engineering*, 1999. to appear.

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-1998-016, Rutherford Appleton Laboratory, 1998.

J.K. Reid and J.A. Scott. Reversing the row order for the row-by-row frontal solver. Technical Report RAL-TR-1999-037, Rutherford Appleton Laboratory, 1999.

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-1998-041, Rutherford Appleton Laboratory, 1997.

J.A. Scott. A new row ordering strategy for frontal solvers. Technical Report RAL-TR-1998-056, Rutherford Appleton Laboratory, 1998. To appear in *Numerical Linear Algebra and Applications*.

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

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

J.A. Vegeais and M.A. Stadtherr. Vector processing strategies for chemical process flowsheeting. *AIChE J.*, **36**, 1687–1696, 1990.

S.E. Zitney and M.A. Stadtherr. Frontal algorithms for equation-based chemical process flowsheeting on vector and parallel computers. *Comput. Chem. Engng.*, **17**, 319–338, 1993.

S.E. Zitney, L. Brull, L. Lang, and R. Zellet. Plantwide dynamic simulation on supercomputers. *AIChE Symp. Ser.*, **91**, 313–316, 1995.

S.E. Zitney, J.U. Mallya, T.A. Davis, and M.A. Stadtherr. Multifrontal vs frontal techniques for chemical process simulation on supercomputers. *Comput. Chem. Engng.*, **20**, 614–646, 1996.