RAL-TR-2005-001

CCLRC

Reducing the total bandwidth of a sparse unsymmetric matrix

J. K. Reid and J. A. Scott

March 2005

Council for the Central Laboratory of the Research Councils

Library and Information Services
CCLRC Rutherford Appleton Laboratory
Chilton Didcot
Oxfordshire OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk

CCLRC reports are available online at:
http://www.clrc.ac.uk/Activity/ACTIVITY=Publications;SECTION=225;

ISSN 1358-6254

RAL-TR-2005-001

Reducing the total bandwidth of a sparse unsymmetric matrix[1,2]

by

J. K. Reid and J. A. Scott

Abstract

For a sparse symmetric matrix, there has been much attention given to algorithms for reducing the bandwidth. As far as we can see, little has been done for the unsymmetric matrix A, which has distinct lower and upper bandwidths l and u. When Gaussian elimination with row interchanges is applied, the lower bandwidth is unaltered while the upper bandwidth becomes $l + u$. With column interchanges, the upper bandwidth is unaltered while the lower bandwidth becomes $l+u$. We therefore seek to reduce $\min(l, u) + l + u$, which we call the **total** bandwidth.

We consider applying the reverse Cuthill-McKee algorithm to $A+A^T$, to the row graph of A, and to the bipartite graph of A. We also propose a variation that may be applied directly to A.

When solving linear systems, if the matrix is preordered to block triangular form, it suffices to apply the band-reducing method to the blocks on the diagonal. We have found that this is very beneficial on matrices from actual applications.

Finally, we have adapted the node-centroid and hill-climbing ideas of Lim, Rodrigues and Xiao to the unsymmetric case and found that these give further worthwhile gains.

Numerical results for a range of practical problems are presented and comparisons made with other possibilities, including the recent lexicographical method of Baumann, Fleishmann and Mutzbauer.

Keywords: matrix bandwidth, sparse unsymmetric matrices, Gaussian elimination.

---

[1] Current reports available from "http://www.numerical.rl.ac.uk/reports/reports.html".

[2] The work of the second author was supported by the EPSRC grant GR/S42170.

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

April 25, 2005.

1 Introduction

If Gaussian elimination is applied without interchanges to an unsymmetric matrix $A = \{a_{ij}\}$ of order n, each fill-in takes place between the first entry of a row and the diagonal or between the first entry of a column and the diagonal. It is therefore sufficient to store all the entries in the lower triangle from the first entry in each row to the diagonal and all the entries in the upper triangle from the first entry in each column to the diagonal. This simple structure allows straightforward code using static data structures to be written. We will call the sum of the lengths of the rows the **lower profile** and the sum of the lengths of the columns the **upper profile**.

We will also use the term **lower bandwidth** for $l = \max_{a_{ij}\neq 0}(i-j)$ and the term **upper bandwidth** for $u = \max_{a_{ij}\neq 0}(j-i)$. For a symmetric matrix, these are the same and are called the **semi-bandwidth**. A particularly simple data structure is available by taking account only of the bandwidths and has the merit that the structure remains after the application of row interchanges, though the upper bandwidth increases to $l+u$. With column interchanges, the upper bandwidth is unaltered while the lower bandwidth becomes $l+u$. We therefore seek to reduce $\min(l,u)+l+u$, which we call the **total bandwidth**. For a band linear equation solver to be efficient, the matrix needs to be preordered so that its total bandwidth is small. In this paper, we consider algorithms for reducing the total bandwidths of matrices with unsymmetric sparsity patterns. We consider band methods rather than profile methods because the upper profile alters when row interchanges are applied and the lower profile alters when column interchanges are applied.

Many algorithms for reducing the bandwidth of a sparse symmetric matrix A have been proposed and most make extensive use of the adjacency graph of the matrix. This has a node for each row (or column) of the matrix and node i is a neighbour of node j if a_{ij} (and by symmetry a_{ji}) is an entry (nonzero) of A. An important and well-known example of an algorithm that uses the adjacency graph is that of Cuthill and McKee (1969), which orders the nodes of the adjacency graph by increasing distance from a chosen starting node. Ordering the nodes in this way groups them into 'level sets', that is, nodes at the same distance from the starting node. Since nodes in level set l can have neighbours only in level sets $l-1$, l, and $l+1$, the reordered matrix is block tridiagonal with blocks corresponding to the level sets. It is therefore desirable that the level sets be small, which is likely if there are many of them. The number of level sets is dependent on the choice of starting node. Therefore, algorithms for finding a good starting node are usually based on finding a pseudo-diameter (pair of nodes that are a maximum distance apart or nearly so), see for example Gibbs, Poole and Stockmeyer (1976) and Reid and Scott (1999) and the references therein. George (1971) found that the profile may be reduced if the Cuthill-McKee ordering is reversed. The reverse Cuthill-McKee (RCM) algorithm and variants of it remain in common use. For example, an implementation is available within MATLAB as the function `symrcm` and RCM is included as an option within the package `mc60` from the mathematical software library HSL (2004).

In this paper, we consider how variants of the Cuthill-McKee algorithm can be used to order unsymmetric matrices for small total bandwidths. In Section 2, we discuss three undirected graphs that can be associated with an unsymmetric matrix A. We then propose in Section 3 an unsymmetric variant of RCM. In Section 4, we look at using a modified version of the hill-climbing algorithm of Lim, Rodrigues and Xiao (2004) to improve a given ordering and, in Section 5, we

propose a variant of the node centroid algorithm of Lim et al. (2004) for the unsymmetric case. In Section 6, we consider permuting A to block triangular form and applying the band reducing algorithms to the diagonal blocks. In Section 7, we examine the effect of row interchanges. Numerical results for a range of practical problems are presented in Section 8.

We end this section by briefly discussing a recently published algorithm for reducing the bandwidth of an unsymmetric matrix. Baumann, Fleishmann and Mutzbauer (2003) reduce the lower and upper bandwidths and profiles by making the pattern of each row and column define a binary number, then alternate between ordering the rows in decreasing order and ordering the columns in decreasing order. They show that this converges to a limit and call it a 'double ordering'. Since only the leading entries of the rows or columns affect the bandwidths and profiles, we have implemented an efficient variant in which no attempt is made to order the rows or columns with the same leading entry. We call this a **relaxed double ordering** (RDO) and include results for it in Section 8.

Unfortunately, there are huge numbers of double orderings and Baumann et al. (2003) have no strategy for choosing a good one. For example, a Cuthill-McKee ordering produces a relaxed double ordering regardless of the starting node, since the leading entries of the rows (or columns) form a monotonic sequence. There is scope for the double ordering to reduce the profile of an RCM ordering, but our experience is that the improvement is slight and is often at the expense of the bandwidths (see Section 8.2).

2 Undirected graphs for unsymmetric matrices

In this section, we consider three adjacency graphs that can be associated with an unsymmetric matrix A. In each case, RCM (reverse Cuthill-McKee) will be employed to reduce the semi-bandwidth of the graph and this permutation will be used to reorder A.

2.1 Using $A + A^T$

For a matrix whose structure is nearly symmetric, an effective strategy is to find a symmetric permutation that reduces the bandwidth of the structure of the symmetric matrix $A + A^T$. The MATLAB function `symrcm` applies RCM to the adjacency graph of $A + A^T$. If the symmetric permutation is applied to A, the lower and upper bandwidths are no greater than the semi-bandwidth of the permuted $A + A^T$. Of course, the same algorithm may be applied to a matrix that is far from symmetric and the same results apply, but the effectiveness is uncertain. It is likely to be helpful to permute A to make it more symmetric. We will judge this by its **symmetry index**, which is the number of off-diagonal entries a_{ij} for which a_{ji} is also an entry divided by the total number of off-diagonal entries. Permuting a large number of off-diagonal entries onto the diagonal reduces the number of unmatched off-diagonal entries, which in turn generally increases the symmetry index (see, for example, Duff and Koster, 1999 and Hu and Scott, 2005). The HSL routine `mc21` can be used to compute a matching that corresponds to a row permutation of A that puts entries onto the diagonal.

2.2 Bipartite graph

The bipartite graph of A has a node for each row and a node for each column and row node i is connected to column node j if a_{ij} is a entry. It is straightforward to see that this is actually the adjacency graph of the $2n \times 2n$ symmetric matrix

$$\hat{A} = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}. \tag{2.1}$$

Starting the Cuthill-McKee algorithm with any node, the level sets are alternately sets of rows and sets of columns. If we start from a row node and perform the corresponding symmetric permutation on the matrix (2.1), we find the matrix

$$\begin{bmatrix} 0 & A_{11} & & & & & & \\ A_{11}^T & 0 & A_{21}^T & & & & & \\ & A_{21} & 0 & A_{22} & & & & \\ & & A_{22}^T & 0 & A_{23}^T & & & \\ & & & A_{32} & 0 & A_{33} & & \\ & & & & A_{33}^T & 0 & A_{34}^T & \\ & & & & & \cdots & \cdots & \cdots \end{bmatrix} \tag{2.2}$$

where A_{lm} is the submatrix of A corresponding to the rows of row level set l and columns of column level set m.

If we permute the rows of A by the row level sets and the orderings within them, and permute the columns by the column level sets and the orderings within them, we find the block bidiagonal form

$$\begin{bmatrix} A_{11} & & & & \\ A_{21} & A_{22} & & & \\ & A_{32} & A_{33} & & \\ & & A_{43} & A_{44} & \\ & & & \cdots & \cdots \end{bmatrix}, \tag{2.3}$$

which is also the submatrix of (2.2) consisting of block rows 1, 3, ... and block columns 2, 4, ... We illustrate with a small reordered example in Figure 2.1. Here there are four row level sets, of sizes 1, 3, 2, 2, and three column level sets, of sizes 3, 2, 3.

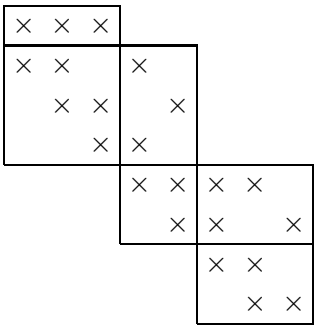

Figure 2.1: A matrix with rows and columns reordered using the permutations obtained by applying Cuthill-McKee to its bipartite graph.

This example has entries on the whole of the diagonal, which will not necessarily be the case. However, if the matrix is structurally nonsingular, the diagonal will always intersect each of the blocks. This is because the leading k columns must be of full structural rank and similarly for the leading k rows.

The semi-bandwidth of the reordered matrix (2.2) is at most one less than the largest sum of the sizes of two adjacent level sets, that is, one less than the largest sum of the sizes of a row level set and an adjacent column level set. The corresponding results for the reordered unsymmetric matrix (2.3) are that the lower bandwidth is at most one less than the sum of the sizes of two adjacent column level sets and the upper bandwidth is at most one less than the sum of the sizes of two adjacent row level sets. Note, however, that all these bounds are pessimistic; they do not take account of the ordering of the nodes within each level set (and RCM does well in this respect) and, in the case (2.3), of the position of the matrix diagonal within the blocks.

2.3 Row graph

Another alternative is to consider the **row graph** (Mayoh, 1965) of A, which is defined to be the adjacency graph of the symmetric matrix AA^T, where matrix multiplication is performed without taking cancellations into account (so that, if a coefficient of AA^T is zero as a result of numerical cancellation, it is still considered to be an entry). The nodes of the row graph correspond to the rows of A and nodes i and j ($i \neq j$) are neighbours if and only if there is at least one column k of A for which a_{ik} and a_{jk} are both entries. The row graph has been used by Scott (1999, 2000) to order the rows of unsymmetric matrices prior to solving the linear system using a frontal solver. We can obtain an ordering for the rows of A by applying the RCM algorithm to AA^T. This will ensure that rows with entries in common are nearby, that is, the first and last entry of each column will not be too far apart. If the columns are now ordered according to their last entry, the lower bandwidth will be small and the upper bandwidth will not be large.

A potential disadvantage of computing and working with the pattern of AA^T is that it can be costly in terms of time and memory requirements. This is because AA^T may contain many more entries than A. It fails completely if A has a full column (AA^T is full), but such a matrix cannot be permuted to have small lower and upper bandwidths.

3 Unsymmetric RCM

Any reordering within a Cuthill-McKee level set of Section 2.2 will alter the positions of the leading entries of the columns of a submatrix A_{ii} or the rows of a submatrix A_{ji}, $j = i + 1$. It will make exactly the same change to the profile of the matrix (2.2) as it does to the sum of the upper and lower profiles of the matrix (2.3). If it reduces the bandwidth of the matrix (2.2), it will reduce either the upper or lower bandwidth of the matrix (2.3); however, the converse is not true: it might reduce the upper or lower bandwidth of the matrix (2.3) without reducing the bandwidth of the matrix (2.2). It follows that it may be advantageous for bandwidth reduction to develop a special-purpose code for the unsymmetric case, rather than giving the matrix (2.1) to a general-purpose code such as `mc60` for reducing the bandwidth of a symmetric matrix. We have developed a prototype unsymmetric bandwidth reduction code of this kind. Again the level sets are alternately sets of rows and set of columns but our unsymmetric Cuthill-McKee

algorithm bases its decisions on the total bandwidth of the unsymmetric matrix A instead of on the bandwidth of the matrix (2.2); results are included in Section 8.2, see Tables 8.4 and 8.5.

The profile and bandwidth are likely to be reduced if the rows of each level set are ordered according to their leading entries. This happens automatically with the Cuthill-McKee algorithm since each level set l is formed by taking the nodes that have not already been included in a level set and are neighbours of the first node of level set $l-1$, then those that have not already been included in a level set and are neighbours of the second node of level set $l-1$, and so on. This was done for the example in Figure 2.1.

Reversing the Cuthill-McKee ordering may reduce the profile. It is reduced if the column index of the trailing entry in a row is lower than the column index of the trailing entry in an earlier row. There is an example in block A_{22} of Figure 2.1.

4 Hill-climbing to improve a given ordering

In this and the next section, we consider algorithms that are not based on level sets in a graph and are therefore completely different.

Lim et al. (2004) propose a hill-climbing algorithm for reducing the semi-bandwidth of a symmetric matrix. An entry a_{ij} in a reordered matrix A with semi-bandwidth b is called **critical** if $|i-j| = b$. For each critical entry a_{ij} in the lower-triangular part, an interchange of i with $k < i$ or j with $k > j$ is sought that will reduce the number of critical entries. For example, a_{94} is critical in Figure 4.2 and the semi-bandwidth is reduced from 5 to 4 by interchanging column

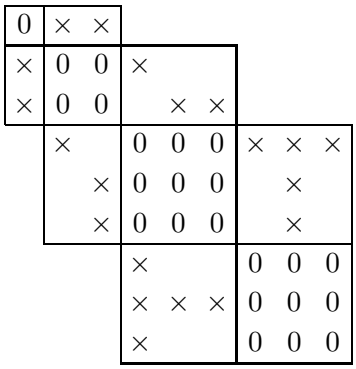

Figure 4.2: A symmetric matrix ordered by Cuthill-McKee.

4 with column 5 and row 4 with row 5. As a column is moved backwards, its first entry is moved away from the diagonal while its last entry is moved nearer. If the distance of the first entry from the diagonal is d, we can therefore limit the choice of k to the range $j < k < j + b - d$. Similarly, if the distance of the last entry in row i from the diagonal is l, we limit the choice of k to the range $i - b + l < k < i$. Each interchange while the semi-bandwidth is b reduces the number of critical entries by one. If the number of critical entries becomes zero, we recommence the algorithm for semi-bandwidth $b - 1$ and continue until none of the critical entries for the current semi-bandwidth can be interchanged to reduce their number. Note that this hill-climbing algorithm cannot increase the semi-bandwidth.

We have adapted this idea to reduce the lower and upper bandwidths of an unsymmetric matrix. If the lower bandwidth is l and the upper bandwidth is u, we call an entry a_{ij} in the

lower triangle for which $i - j = l$ a **critical lower** entry and an entry a_{ij} in the upper triangle for which $j - i = u$ a **critical upper** entry. We have found it convenient to alternate between making row interchanges while the column permutation is fixed and making column interchanges while the row permutation is fixed. While making row interchanges to reduce the number of critical upper entries, we seek to exchange a row i containing a critical upper entry with another row so that the number of critical upper entries is reduced by one while the lower bandwidth is not increased. If the distance between the leading entry in the row and the diagonal is d, we limit our search to rows in the range $i - l + d \leq k < i$. For example, we do not exchange rows 3 and 4 in Figure 2.1 since this would increase the lower bandwidth.

Similarly, while making row interchanges to reduce the number of critical lower entries, we seek to exchange a row i containing a critical lower entry with another row so that the number of critical lower entries is reduced by one while the upper bandwidth is not increased.

One complete iteration of our hill-climbing algorithm consists of row hill-climbing (using row interchanges to first reduce the lower bandwidth as much as possible and then to reduce the upper bandwidth as much as possible), followed by column hill-climbing (using column interchanges to first reduce the upper bandwidth as much as possible and then to reduce the lower bandwidth as much as possible). We continue until a complete iteration fails to reduce one of the bandwidths or the total number of critical entries.

5 Node centroid ordering

The hill-climbing algorithm of the previous section is essentially a local search and is very dependent on the initial order that it is given. To generate other initial orderings, Lim et al. (2004) propose an algorithm that they call 'node centroid'. For the graph of a symmetric matrix, they define $N_\lambda(i)$ to the set of neighbours j of node i for which $|i-j|$ is at least λb, where b is the bandwidth and $\lambda \leq 1$ is a parameter for which they recommend a value of 0.95. They refer to such neighbours as **λ-critical**. $w(i)$ is then defined as the average node index over $i \cup N_\lambda(i)$ and the nodes are ordered by increasing $w(i)$. This will tend to move a row with a λ-critical entry in the lower triangle but no λ-critical entry in the upper triangle forward; hopefully, its new leading entry will be nearer the diagonal than the old one was and its trailing entry will not have moved out so much that it becomes critical. Similar arguments apply to a row with a λ-critical entry in the upper triangle but no λ-critical entry in the lower triangle, which will tend to be moved back.

Lim et al. (2004) apply a sequence of major steps, each of which consists of two iterations of node centroid ordering followed by one iteration of hill-climbing and report encouraging results for the DWT set of symmetric problems from the Harwell-Boeing Sparse Matrix Collection (Duff, Grimes and Lewis, 1989).

We have adapted this idea to the unsymmetric case by again alternating between permuting the rows while the column permutation is fixed and permuting the columns while the row permutation is fixed. Suppose the lower bandwidth is l and the upper bandwidth is u. While permuting the rows, only the leading and trailing entries of the rows are relevant since they will still have these properties after the row permutation. If the leading or trailing entry of row i is λ-critical it is desirable to move the row. If its leading entry is in column l_i and its trailing entry is in column u_i, the gap between the upper band and the trailing entry is $u + i - u_i$ and the gap

between the lower band and the leading entry is $l_i - (i - l) = l_i - i + l$. If we move the row forward to become row $i + \delta$, the gaps become $u + i + \delta - u_i$ and $l_i - i - \delta + l$. If $l > u$, it would seem desirable to make the gap at the trailing end greater than the gap at the leading end. We choose a parameter $\alpha > 1$ and aim for the gap at the trailing end to be α times greater than the gap at the leading end, that is,

$$u + i + \delta - u_i = \alpha(l_i - i - \delta + l), \tag{5.4}$$

or

$$\delta = \frac{(u_i - i - u) + \alpha(l_i - i + l)}{1 + \alpha}. \tag{5.5}$$

Similar calculations for $l = u$ and $l < u$ lead us to conclude that a desirable position for the row is given by the equation

$$w(i) = \begin{cases} i + \frac{(u_i - i - u) + \alpha(l_i - i + l)}{1 + \alpha} & \text{if } l > u, \\ i + \frac{(u_i - i - u) + (l_i - i + l)}{2} & \text{if } l = u, \\ i + \frac{\alpha(u_i - i - u) + (l_i - i + l)}{1 + \alpha} & \text{if } l < u. \end{cases} \tag{5.6}$$

For other rows, we set $w(i) = i$. We sort the rows in increasing order of $w(i), i = 1, 2, \ldots, n$. In our numerical experiments (see Section 8), we found that a suitable value for α is 2.

Similar considerations apply to ordering the columns with the row order fixed. We apply a sequence of up to ten major steps, each consisting of two iterations of the node centroid row ordering followed by row hill-climbing, then two iterations of the node centroid column ordering followed by column hill-climbing. We continue while the total bandwidth ceases to decrease.

6 The block triangular form

In the symmetric case, it may be possible to preorder the matrix A to block diagonal form

$$\begin{bmatrix} A_{11} & & & & & \\ & A_{22} & & & & \\ & & A_{33} & & & \\ & & & A_{44} & & \\ & & & & \cdots & \end{bmatrix}. \tag{6.7}$$

In this case, each block may be permuted to band form and the overall matrix is a band matrix; the profile is the sum of the profiles of the blocks and the bandwidth is the greatest bandwidth of a block.

The unsymmetric case is not so straightforward because we need also to exploit the block triangular form

$$\begin{bmatrix} A_{11} & & & & & \\ A_{21} & A_{22} & & & & \\ A_{31} & A_{32} & A_{33} & & & \\ A_{41} & A_{42} & A_{43} & A_{44} & & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \end{bmatrix}, \tag{6.8}$$

where the blocks A_{ll}, $l = 1, 2, ..., N$, are all square. A matrix that can be permuted to this form with $N > 1$ diagonal blocks is said to be **reducible**; if no block triangular form other than the

trivial one with a single block ($N = 1$) can be found, the matrix is **irreducible**. The advantage of the block triangular form (6.8) is that the corresponding set of equations $Ax = b$ may be solved by the block forward substitution

$$A_{ii}x_i = b_i - \sum_{j=1}^{i-1} A_{ij}x_j, \ i = 1, 2, \ldots, N. \tag{6.9}$$

There is no fill in the off-diagonal blocks, which are involved only in matrix-by-vector multiplications. It therefore suffices to permute each diagonal block A_{ii} to band form. We will take the upper and lower profiles to be the sums of the upper and lower profiles of the diagonal blocks and the upper and lower bandwidths to be the greatest of the upper and lower bandwidths of the diagonal blocks.

We will assume that each diagonal block has entries on its diagonal since most algorithms for finding the block triangular form begin with a permutation that places entries on the diagonal. Therefore, permuting entries onto the diagonal is not available as a strategy for improving the symmetry index (see Section 2.1).

7 The effect of interchanges

If row interchanges are needed for stability reasons during the factorization of an unsymmetric matrix A with lower bandwidth l and upper bandwidth u, it may be readily verified that the lower bandwidth remains l but the upper bandwidth may increase to $l + u$. Thus, we can take advantage of l being less than u. If l is greater than u, we may factorize A^T instead of A or use column interchanges instead of row interchanges. In both cases, one triangular factor has bandwidth $\min(l, u)$ and the other has bandwidth $l + u$. We therefore introduce the term **total bandwidth** for the sum $\min(l, u) + l + u$. For a reducible matrix, we take the total bandwidth to be the greatest total bandwidth of a diagonal block of its block triangular form.

Most band solvers, including `_GBTRF` of LAPACK (Anderson, Bai, Bischof, Blackford, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney and Sorensen, 1999), simply hold explicitly any zeros with the fixed bandwidth form with lower bandwidth l and upper bandwidth $l+u$ and perform explicit operations for them. An exception is the code `ma65` of HSL (2004). This works within the fixed bandwidth form, but avoids many of the operations on explicit zeros. It ignores entries outside a profile defined by the indices of the row and column ends, which are held in the n-vectors `rowend` and `colend`. It requires that the diagonal entries are held (`rowend`$(i) \geq i$, `colend`$(j) \geq j$) and begins by increasing `colend` to be monotonic thus

```
do j = 2,n
   colend(j) = max(colend(j-1),colend(j))
end do
```

This involves little loss of efficiency since the corresponding entries are almost certain to fill-in during factorization. The vector `colend` is static thereafter. The vector `rowend` is not static, however. If two rows are interchanged, the corresponding components of `rowend` need to be interchanged and the row end of each non-pivot row may need to be increased before elimination operations are applied to it. The row operations associated with a pivot are performed one by one, and are skipped if the size of the multiplier is less that a threshold with default value zero.

Thus advantage is taken of zeros in the pivot column, but not of zeros within the pivot row since that would add the overheads of indirect addressing.

8 Numerical experiments

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

8.1 Test problems

Table 8.1: The test problems.

Identifier	Order	Number of entries	Symmetry index
4cols†	11770	43668	0.0159
10cols†	29496	109588	0.0167
bayer01	57735	277774	0.0002
bayer03	6747	56196	0.0031
bayer04	20545	159082	0.0016
bayer09	3083	21216	0.0212
circuit_3	12127	48137	0.7701
ethylene-1†	10673	80904	0.2973
extr1	2837	11407	0.0042
hydr1	5308	23752	0.0041
icomp†	69174	301465	0.0001
impcol_d	425	1339	0.0567
lhr34c	35152	764014	0.0015
lhr71c	70304	1528092	0.0015
poli_large	15575	33074	0.0035
radfr1	1048	13299	0.0537
rdist1	4134	94408	0.0588
rdist2	3198	56934	0.0456
rdist3a	2398	61896	0.1404
Zhao2	33861	166453	0.9225

The test problems are listed in Table 8.1. Each arises from a real engineering or industrial application. Problems marked with a † are chemical process engineering problems that were supplied to us by Mark Stadtherr of the University of Notre Dame. The remaining problems are available through the University of Florida Sparse Matrix Collection (Davis, 1997); further details may be found at `www.cise.ufl.edu/research/sparse/matrices/`. Most of the test problems were chosen on the grounds of being highly unsymmetric because working with the symmetrized matrix $A + A^T$ will be satisfactory for near-symmetric matrices. We include one near-symmetric matrix to illustrate this.

In Table 8.2, we give details of the block triangular form for each of our test matrices. The number of 1×1 and 2×2 blocks (n_1 and n_2) as well as the number of larger blocks ($n_{>2}$) and the total number of entries in the off-diagonal blocks (n_{off}) are given. The size (m) of the largest diagonal block A_{kk}, the number of entries (me) and average number of entries per row ($avg(me)$) in A_{kk} and its symmetry index (si) are given, as well as the number of entries in the matrix

Table 8.2: Details of the block triangular form for our test problems. n_1 and n_2 are the numbers of 1×1 and 2×2 blocks; $n_{>2}$ is the number of larger blocks; n_{off} is the total number of entries in the off-diagonal blocks. For the largest diagonal block A_{kk}, m is the order, me is the number of entries, $avg(me)$ is the average number of entries per row, si is the symmetry index, and $me(A_{kk}A_{kk}^T)$ is the number of entries in $A_{kk}A_{kk}^T$.

Identifier	n_1	n_2	$n_{>2}$	n_{off}	Largest block A_{kk}				
					m	me	$avg(me)$	si	$me(A_{kk}A_{kk}^T)$
4cols	0	0	1	0	11770	43668	3.71	0.0159	210026
10cols	0	0	1	0	29496	109588	3.72	0.0167	527124
bayer01	8858	0	3	28228	48803	240222	4.92	0.0812	1236678
bayer03	1772	2	6	19575	4776	33555	7.02	0.1066	252236
bayer04	6349	19	10	56096	13762	93750	6.81	0.0976	690982
bayer09	1466	2	7	8476	1364	9562	7.01	0.0808	71842
circuit_3	4520	0	1	9593	7607	34024	4.47	0.5579	76178
ethylene-1	2137	0	7	12865	8336	65375	7.84	0.3000	203920
extr1	424	0	1	464	2413	10519	4.35	0.0935	34118
hydr1	968	0	6	1420	2370	11738	4.95	0.0730	47946
icomp	44988	2	2061	215880	185	741	4.00	0.1187	2554
impcol_d	226	0	1	551	199	562	2.82	0.0275	1350
lhr34c	3519	0	14	47924	7663	173683	22.7	0.3982	7733752
lhr71c	7038	0	28	95912	7663	173683	22.7	0.4225	773752
poli_large	15450	12	4	17266	90	286	3.18	0.1633	680
radfr1	97	0	1	969	951	12233	12.9	0.4751	34032
rdist1	198	0	1	3959	3936	90251	22.9	0.4821	280538
rdist2	198	0	1	2968	3000	53768	17.9	0.4868	164284
rdist3a	98	0	1	2252	2300	59546	25.9	0.4500	188612
Zhao2	0	0	1	0	33861	166453	4.92	0.9225	549692

$A_{kk}A_{kk}^T$. We note that 4cols, 10cols, and Zhao2 are irreducible while a number of problems (including circuit_3 and the rdist examples) have only one block of order greater than 1. Most of the remaining problems have fewer than 10 blocks of order greater than 1; the main exception is icomp, which has 2061 such blocks of which 8 are of size over 30. As expected, the matrix $A_{kk}A_{kk}^T$ contains many more entries than A_{kk}. We also note that, for the reducible examples, the symmetry index of A_{kk} is usually larger than that of the original matrix.

8.2 Test results

We first present results for applying the mc60 implementation of the RCM algorithm to the following matrices: (*i*) $A + A^T$, (*ii*) $B + B^T$, where $B = PA$ is the permuted matrix after employing mc21 to put entries on the diagonal, (*iii*) AA^T and (*iv*) the matrix $\hat{A}$ given by (2.1). In each case, the ordering is then applied to A (as described in Section 2). The total bandwidth (defined in Section 7) for each ordering and for the initial ordering is given in Table 8.3. Results are also given for the relaxed double ordering (RDO), see end of Section 1. A blank entry in the $B+B^T$ column indicates the matrix A has no zeros on the diagonal and, in these cases, mc21 is not applied. We see that applying mc21 prior to the reordering with RCM can significantly reduce the bandwidths but narrower bandwidths are achieved by working with either the row graph (AA^T) or the bipartite graph ($\hat{A}$). For the majority of our test examples, the RDO orderings are much

Table 8.3: The total bandwidth for the RDO and RCM ordering algorithms.

Identifier	Initial	RDO	RCM			
			$A + A^T$	$B + B^T$	AA^T	$\hat{A}$
4cols	13305	4768	846		460	565
10cols	30532	13855	1052		546	572
bayer01	157073	45332	52201	4117	2232	2236
bayer03	17744	3660	7873	1074	651	651
bayer04	61436	14880	18502	7133	3813	3902
bayer09	8796	1977	3124	688	427	430
circuit_3	36231	10979	17658	20584	10441	11157
ethylene-1	10664	8301	7797		5114	5093
extr1	7798	1610	2575	298	171	169
hydr1	14377	2726	5800	559	337	334
icomp	56862	43801	1429		1262	1262
impcol_d	496	153	241	219	123	117
lhr34c	57141	13924	27428	5332	3296	3771
lhr71c	58291	18181	27064	5802	3620	3771
poli_large	46466	6400	16849		6381	6316
radfr1a	1200	95	970	142	71	98
rdist1	4446	195	3421	336	223	215
rdist2	3470	146	2380	370	169	165
rdist3a	2610	229	1858	570	225	250
Zhao2	86377	34409	1476		1464	1476

poorer, although they are a significant improvement on the initial ordering.

Table 8.4 shows the effect of applying ordering algorithms to the diagonal blocks of the block triangular form (6.8). As already noted, the construction of the block triangular form ensures that there are no zeros on the diagonal, so we do not preorder using `mc21`. Apart from this, the algorithms featured in Table 8.3 are featured here too. We have experimented with running RDO after RCM applied to AA^T; the results are in parentheses in column 6 of Table 8.4. Column 8 shows the result of applying our unsymmetric RCM code (see Section 3) to the diagonal blocks. We have highlighted the narrowest bands and those within 3 per cent of the narrowest. This margin is based on our experience with all the algorithms that an individual result quite often changes of about 3 per cent if we randomly permute the rows and columns of A before applying the algorithm. As expected, the larger symmetry index for the diagonal blocks of the block triangular form results in an improvement in the performance of RCM applied to $A + A^T$ but in all cases it is better to use the other RCM variants. There appears to be little to choose between RCM applied to the row graph, RCM applied to the bipartite graph, and our unsymmetric RCM algorithm; for some of the examples, each produces the narrowest total bandwidth. Although RDO does improve the bandwidths for a number of problems (including `icomp` and `poli_large`), for others the results are significantly worse (for example, `4cols` and `bayer01`) and so we do not recommend its use.

So far, the reported results have not included hill climbing. In Table 8.5 we present results for applying the different RCM variants to the block triangular form followed by applying both hill climbing alone (denoted by RCM + HC) and the node centroid algorithm plus hill climbing (denoted by RCM + NC + HC). For the node centroid algorithm we have experimented with

Table 8.4: The total bandwidth for the RDO and RCM ordering algorithms applied to the diagonal blocks of the block triangular form.

Identifier	Initial	RDO	RCM				
			$A+A^T$	AA^T	(+RDO)	$\hat{A}$	A
4cols	13305	4768	846	**460**	(1001)	565	504
10cols	30532	13855	1052	546	(1600)	572	**528**
bayer01	146387	34581	3483	**1768**	(3056)	1823	**1776**
bayer03	14314	3617	740	527	(547)	**500**	**506**
bayer04	41245	9019	1746	1003	(1846)	**939**	**944**
bayer09	4039	529	385	279	(433)	**248**	275
circuit_3	22795	4776	1903	**1330**	(1394)	**1321**	**1297**
ethylene-1	24996	4967	323	**179**	(432)	**184**	230
extr1	7211	1660	240	**145**	(266)	**149**	**148**
hydr1	7068	1640	198	129	(**112**)	134	129
icomp	536	148	183	151	(**140**)	153	**140**
impcol_d	582	70	98	79	(82)	67	**59**
lhr34c	22984	4397	982	**669**	(2171)	720	721
lhr71c	22972	5135	991	741	(2173)	**727**	**720**
poli_large	256	84	97	91	(**79**)	84	**77**
radfr1a	621	132	130	88	(93)	**85**	93
rdist1	341	155	346	**188**	(**193**)	**189**	**192**
rdist2	267	**103**	276	121	(143)	117	120
rdist3a	6888	1789	293	192	(186)	**168**	185
Zhao2	86377	34409	1471	**1454**	(2196)	1467	**1424**

using values of λ in the range $[0.8, 1]$ and values of α in the range $[1.5, 2.5]$. Our experience was that the results were not very sensitive to the precise choice of λ and for most examples 0.85 gave results that were within 3 per cent of the best. For α, we found that a value of 2 gave slightly better results that either 1.5 or 2.5. We therefore used $\lambda = 0.85$ and $\alpha = 2$ for the results in Table 8.5.

Again, the narrowest total bandwidths (and those within 3 per cent of the narrowest) are highlighted. Comparing the results in columns 2 to 5 of Table 8.5 with the corresponding results in Table 8.4, we see that hill climbing (which never increases the total bandwidth) can lead to some significant improvements. For some problems, for example `bayer09`, the savings are over 20 per cent. However, looking also at columns 6 to 9, it is clear that for all but problem `Zhao2`, the smallest bandwidths are achieved by using RCM combined with both the node centroid and the hill-climbing algorithms. The largest improvements resulting from using the node centroid algorithm are to the orderings obtained using RCM applied to $A + A^T$; for some problems (including the `bayer` and the `lhr` examples) the reductions resulting from including the node centroid algorithm are more than 30 per cent. However, for many of our examples, one of the other variants generally produces orderings with a smaller total bandwidth.

8.3 Comparisons with a general sparse solver

We end this section by reporting on using our band-reducing orderings with the HSL band solver `MA65`. Comparisons are made with the general-purpose sparse direct solver `MA48`. `MA48` uses sparse Gaussian elimination for solving unsymmetric systems. During the last decade it has

Table 8.5: The total bandwidth after hill climbing and the node centroid. RCM + HC denotes RCM followed by hill climbing; RCM + HC + NC denotes RCM followed by hill climbing and the node centroid algorithm. All are applied to the diagonal blocks of the block triangular form.

Identifier	RCM + HC				RCM + NC + HC			
	$A+A^T$	AA^T	$\hat{A}$	A	$A+A^T$	AA^T	$\hat{A}$	A
4cols	718	435	549	481	502	**395**	458	443
10cols	902	498	553	479	625	**448**	462	**447**
bayer01	3241	1739	1742	1756	2243	**1659**	**1675**	**1659**
bayer03	668	446	445	452	411	**381**	**384**	**377**
bayer04	1507	899	917	868	941	856	**822**	**809**
bayer09	276	206	212	215	184	**162**	173	**167**
circuit_3	1715	1228	1227	1123	1356	**1065**	**1074**	**1095**
ethylene-1	271	172	173	216	174	169	**162**	203
extr1	190	119	120	131	130	**115**	119	**116**
hydr1	133	101	101	120	**89**	**91**	**91**	**89**
icomp	93	139	130	84	89	93	**72**	**75**
impcol_d	66	61	56	55	**50**	51	**49**	52
lhr34c	850	626	591	601	546	558	**528**	**533**
lhr71c	862	626	598	576	**540**	572	557	**540**
poli_large	56	70	70	66	54	**50**	61	**52**
radfr1a	**57**	63	72	76	**58**	**58**	**57**	**58**
rdist1	148	133	156	158	**123**	**121**	**124**	**119**
rdist2	112	93	111	120	92	**89**	**90**	**88**
rdist3a	155	160	161	184	**139**	**138**	**139**	**139**
Zhao2	1471	1454	1467	**1420**	1473	**1446**	**1462**	**1442**

been very widely used and has been incorporated into a number of commercial packages; indeed, it has become a benchmark against which other sparse direct solvers are frequently compared. The analyse phase of MA48 first permutes the matrix to block triangular form and then, for each submatrix of the block diagonal, selects a pivot sequence using a Markowitz criterion for maintaining sparsity and threshold partial pivoting for numerical stability. This pivot sequence is then passed to the factorization phase. A number of factorizations may follow a single call to the analyse phase. Full details are given in Duff and Reid (1996).

In Table 8.6 we present times for the factorization phases of MA65 and MA48. MA65 is used with our unsymmetric RCM algorithm followed by the node centroid algorithm plus hill climbing (see final column of Table 8.5). The experiments were performed on a single Xeon 3.06 GHz processor of a Dell Precision Workstation 650 with 4 GBytes of RAM. The NAG Fortran 95 compiler was used with the compiler optimization flag -O. All reported timings are CPU times, measured using the Fortran 95 routine cpu_time and are given in seconds. For the problems with a factorization time of less than 1 second, the factorization phase was called repeatedly until the accumulated time was at least 1 second; the average factorization time is reported. We see that, for all the problems in the top half of the table, MA48 is significantly faster than MA65. However, for many of the problems below impcol_d, MA65 is the faster code. Looking again at Table 8.2, it appears that MA48 performs very well on the highly unsymmetric and very sparse blocks while MA65 is more suited to factorizing blocks that are denser and have a larger symmetry index.

We note that it is essential to the performance of MA65 that row operations associated with a pivot are skipped if the multiplier is zero. We found that not reversing the unsymmetric Cuthill-

Table 8.6: Factorization times for MA48 and MA65.

Identifier	MA48	MA65
4cols	**0.069**	0.220
10cols	**0.206**	0.610
bayer01	**0.447**	4.330
bayer03	**0.030**	0.103
bayer04	**0.473**	0.985
bayer09	**0.004**	0.010
circuit_3	**0.008**	0.298
ethylene-1	**0.033**	0.089
extr1	**0.003**	0.010
hydr1	**0.011**	0.016
impcol_d	**0.0002**	0.0003
lhr34c	1.150	**1.120**
lhr71c	**2.260**	2.390
poli_large	**0.001**	0.007
radfr1a	0.004	**0.003**
rdist1	0.093	**0.049**
rdist2	0.038	**0.023**
rdist3a	0.056	**0.025**
Zhao2	108.8	**69.8**

McKee ordering has little effect on the total bandwidth but can lead to much less skipping and hence to a significant increase in the factorization time and operation count for MA65 (for some of our problems the increase was greater than a factor of 10).

9 Concluding remarks

We have considered algorithms for reducing the lower and upper bandwidths l and u of an unsymmetric matrix A, focusing on the total bandwidth, which we have defined as $l+u+\min(l,u)$, because this is relevant for the storage and work when sets of banded linear equations are solved by Gaussian elimination.

The least satisfactory results came from working with the lexicographical method of Baumann et al. (2003) and with the matrix $A + A^T$. In most cases, they gave inferior results, although the use of the node centroid algorithm plus hill climbing dramatically improved the results of applying reverse Cuthill-McKee ordering to $A + A^T$.

We obtained good results by applying the reverse Cuthill-McKee algorithm to the matrices AA^T (whose graph is the row graph) and $\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$ (whose graph is the bipartite graph). We found similarly good results with an unsymmetric variation of reverse Cuthill-McKee that is applied directly to A.

The results were improved by preordering A to block triangular form and applying one of these three algorithms to the blocks on the diagonal. The rest of the matrix is used unaltered. The bandwidths were further reduced by our unsymmetric node-centroid and hill-climbing algorithms.

We used our final orderings with the HSL variable band solver, MA65, and compared the factorization times with those for the general purpose sparse direct solver MA48. Our results

suggest that for problems that are not highly unsymmetric and not very sparse, using a band solver can be the faster approach.

10 Acknowledgements

We are grateful to Iain Duff of the Rutherford Appleton Laboratory and Yifan Hu of Wolfram Research for helpful comments on a draft of this report.

References

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide, Third Edition*. SIAM, Philadelphia, 1999.

M. Baumann, P. Fleishmann, and O. Mutzbauer. Double ordering and fill-in for the LU factorization. *SIAM J. Matrix Analysis and Applications*, **25**, 630–641, 2003.

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. Full details from www.cise.ufl.edu/research/sparse/matrices/.

I.S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Analysis and Applications*, **20**, 889–901, 1999.

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 Transactions on Mathematical Software*, **22**, 187–226, 1996.

I.S. Duff, R.G. Grimes, and J.G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, **15**, 1–14, 1989.

A. George. Computer implementation of the finite-element method. Report STAN CS-71-208, Ph.D Thesis, Department of Computer Science, Stanford University, 1971.

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.

HSL. A collection of Fortran codes for large-scale scientific computation, 2004. See `http://hsl.rl.ac.uk/`.

Y.F. Hu and J.A. Scott. Ordering techniques for singly bordered block diagonal forms for unsymmetric parallel sparse direct solvers. *Numerical Linear Algebra with Applications*, 2005. To appear.

A. Lim, B. Rodrigues, and F. Xiao. A centroid-based approach to solve the bandwidth minimization problem. *Proceedings of the 37th Hawaii international conference on system sciences, IEEE*, 2004.

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. *Inter. Journal on Numerical Methods in Engineering*, **45**, 1737–1755, 1999.

J.A. Scott. A new row ordering strategy for frontal solvers. *Numerical Linear Algebra with Applications*, **6**, 1–23, 1999.

J.A. Scott. Row ordering for frontal solvers in chemical process engineering. *Computers in Chemical Engineering*, **24**, 1865–1880, 2000.