HH

Technical Report
RAL-TR-97-059

The Design and Use of Algorithms for Permuting Large Entries to the Diagonal of Sparse Matrices

I S Duff and J Koster

November 1997

COUNCIL FOR THE CENTRAL LABORATORY OF THE RESEARCH COUNCILS

Enquiries about copyright, reproduction and requests for additional copies of this report should be addressed to:

The Central Laboratory of the Research Councils
Library and Information Services
Rutherford Appleton Laboratory
Chilton
Didcot
Oxfordshire
OX11 0QX
Tel: 01235 445384 Fax: 01235 446403
E-mail library@rl.ac.uk

ISSN 1358-6254

RAL-TR-97-059

The design and use of algorithms for permuting large entries to the diagonal of sparse matrices[1]

Iain S. Duff[2] and Jacko Koster[3]

ABSTRACT

We consider techniques for permuting a sparse matrix so that the diagonal of the permuted matrix has entries of large absolute value. We discuss various criteria for this and consider their implementation as computer codes. We then indicate several cases where such a permutation can be useful. These include the solution of sparse equations by a direct method and by an iterative technique. We also consider its use in generating a preconditioner for an iterative method. We see that the effect of these reorderings can be dramatic although the best *a priori* strategy is by no means clear.

Keywords: sparse matrices, maximum transversal, direct methods, iterative methods, preconditioning.

AMS(MOS) subject classifications: 65F05, 65F50.

---

[1] Current reports available by anonymous ftp from matisa.cc.rl.ac.uk in the directory "pub/reports". This report is in file dukoRAL97059.ps.gz. Also published as Technical Report TR/PA/97/XX from CERFACS.

[2] isd@rl.ac.uk, also at CERFACS.

[3] koster@cerfacs.fr, CERFACS, 42 Ave G Coriolis, 31057 Toulouse Cedex, France.

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

October 24, 1997.

Contents

1 Introduction

We study algorithms for the permutation of a square unsymmetric sparse matrix $\mathbf{A}$ of order n so that the diagonal of the permuted matrix has large entries. This can be useful in several ways. If we wish to solve the system

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

where $\mathbf{A}$ is a nonsingular square matrix of order n and $\mathbf{x}$ and $\mathbf{b}$ are vectors of length n, then a preordering to place large entries on the diagonal can be useful whether direct or iterative methods are used for solution.

For direct methods, putting large entries on the diagonal suggests that pivoting down the diagonal might be more stable. There is, of course, nothing rigorous in this and indeed stability is not guaranteed. However, if we have a solution scheme like the multifrontal method of Duff and Reid (1983), where a symbolic phase chooses the initial pivotal sequence and the subsequent factorization phase then modifies this sequence for stability, it can mean that the modification required is less than if the permutation were not applied.

For iterative methods, simple techniques like Jacobi or Gauss-Seidel converge more quickly if the diagonal entry is large relative to the off-diagonals in its row or column and techniques like block iterative methods can benefit if the entries in the diagonal blocks are large. Additionally, for preconditioning techniques, for example for diagonal preconditioning or incomplete LU preconditioning, it is intuitively evident that large diagonals should be beneficial.

We consider more precisely what we mean by such permutations in Section 2, and we discuss algorithms for performing them and implementation issues in Section 3. We consider the effect of these permutations when using direct methods of solution in Section 4 and their use with iterative methods in Sections 5 and 6, discussing the effect on preconditioning in the latter section. Finally, we consider some of the implications of this current work in Section 7.

Throughout, the symbols $|x|$ should be interpreted in context. If x is a scalar, the modulus is intended; if x is a set, then the cardinality, or number of entries in the set, is understood.

2 Permuting a matrix to have large diagonals

2.1 Transversals and maximum transversals

We say that an $n \times n$ matrix $\mathbf{A}$ has a large diagonal if the absolute value of each diagonal entry is large relative to the absolute values of the off-diagonal entries in its row and column. We will be concerned with permuting the rows and columns of the matrix so the resulting diagonal of the permuted matrix has this property.

That is, for the permuted matrix, we would like the ratio

$$\frac{|a_{jj}|}{\max_{i \neq j} |a_{ij}|} \tag{2.1}$$

to be large for all j, $1 \leq j \leq n$. Of course, it is not even possible to ensure that this ratio is greater than 1.0 for all j as the simple example $\begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$ shows. It is thus necessary to first scale the matrix before computing the permutation. An appropriate scaling would be to scale the columns so that the largest entry in each column is 1.0. The algorithm that we describe in Section 2.2 would then have the effect of maximizing (2.1).

For an arbitrary nonsingular $n \times n$ matrix, it is a necessary and sufficient condition that for a set of n entries to be permuted to the diagonal, no two can be in the same row and no two can be in the same column. Such a set of entries is termed a maximum transversal, a concept that will be central to this paper and which we now define more rigorously.

We let T denote a set of (at most n) ordered index pairs (i, j), $1 \leq i, j \leq n$, in which each row index i and each column index j appears at most once. T is called a *transversal* for matrix $\mathbf{A}$, if $a_{ij} \neq 0$ for each $(i, j) \in T$. T is called a *maximum* transversal if it has largest possible cardinality. $|T|$ is equal to n if the matrix is nonsingular. If indeed $|T| = n$, then T defines an $n \times n$ permutation matrix $\mathbf{P}$ with

$$\begin{cases} p_{ij} = 1, & \text{for } (i, j) \in T, \\ p_{ij} = 0, & \text{otherwise} \end{cases}$$

so that $\mathbf{P}^T\mathbf{A}$ is the matrix with the transversal entries on the diagonal.

In sparse system solution, a major use of transversal algorithms is in the first stage of permuting matrices to block triangular form. The matrix is first permuted by an unsymmetric permutation to make its diagonal zero-free after which a symmetric permutation is used to obtain the block triangular form. An important feature of this approach is that the block triangular form does not depend on which transversal is found in the first stage (Duff 1977). A maximum transversal is also required in the generalization of the block triangular ordering developed by (Pothen and Fan 1990).

2.2 Bottleneck transversals

We will consider two strategies for obtaining a maximum transversal with large transversal entries. The primary strategy that we consider in this paper is to maximize the smallest value on the diagonal of the permuted matrix. That is, we compute a maximum transversal T such that for any other maximum transversal T_1 we have

$$\min_{(i,j) \in T_1} |a_{ij}| \leq \min_{(i,j) \in T} |a_{ij}|.$$

Transversal T is called a *bottleneck transversal*[1], and the smallest value $|a_{ij}|$, $(i,j) \in T$, is called the *bottleneck value* of $\mathbf{A}$. Equivalently, if $|T| = n$, the smallest value on the diagonal of $\mathbf{P}^T\mathbf{A}$ is maximized, over all permutations $\mathbf{P}$, and equals the bottleneck value of $\mathbf{A}$.

An outline of an algorithm that computes a bottleneck transversal T' for a matrix $\mathbf{A}$ is given below. We assume that we already have an algorithm for obtaining a maximum transversal and denote by $MT(A,T)$ a routine that returns a maximum transversal for a matrix $\mathbf{A}$, starting with the initial "guess" transversal T. We let $\mathbf{A}_\epsilon$ denote the matrix that is obtained by setting to zero in $\mathbf{A}$ all entries $|a_{ij}|$ for which $|a_{ij}| < \epsilon$ (thus $\mathbf{A}_0 = \mathbf{A}$) and T_ϵ denote the transversal obtained by removing from transversal T all the elements (i,j) for which $|a_{ij}| < \epsilon$ (thus $T_0 = T$).

ALGORITHM BT
Initialization:
Set ϵmin to zero and ϵmax to infinity.
$M := MT(A, \emptyset)$;
$T' := M$;
while (there exist i,j such that $\epsilon min < |a_{ij}| < \epsilon max$) **do**
begin
choose $\epsilon = |a_{ij}|$;
(We discuss how this is chosen later)
$T := MT(A_\epsilon, T'_\epsilon)$;
if $|T| = |M|$
then
$T' := T$; $\quad (\star)$
$\epsilon min := \epsilon$;
else
$\epsilon max := \epsilon$;
endif
end;
Complete transversal for permutation;
(Needed if matrix structurally singular)

M is a maximum transversal for $\mathbf{A}$, and hence $|M|$ is the required cardinality of the bottleneck transversal T' that is to be computed. If $\mathbf{A}$ is nonsingular, then $|M| = n$. Throughout the algorithm, ϵmax and ϵmin are such that a maximum transversal of size $|M|$ does not exist for $\mathbf{A}_{\epsilon max}$ but does exist for $\mathbf{A}_{\epsilon min}$. At each step, ϵ is chosen in the interval $(\epsilon min, \epsilon max)$, and a maximum transversal for the matrix $\mathbf{A}_\epsilon$ is computed. If this transversal has size $|M|$, then ϵmin is set to ϵ,

[1]The term *bottleneck* has been used for many years in assignment problems, for example (Fulkerson, Glicksberg and Gross 1953)

otherwise ϵmax is set to ϵ. Hence, the size of the interval decreases at each step and ϵ will converge to the bottleneck value. After termination of the algorithm, T' is the computed bottleneck transversal and ϵ the corresponding bottleneck value. The value for ϵ is unique. The bottleneck transversal T' is not usually unique.

Algorithm BT makes use of algorithms for finding a maximum transversal. The currently known algorithm with best asymptotic bound for finding a maximum transversal is by Hopcroft and Karp (1973). It has a worst-case complexity of $\mathcal{O}(\sqrt{n}\tau)$, where τ is the number of entries in the matrix. An efficient implementation of this algorithm can be found in Duff and Wiberg (1988). The depth-first search algorithm implemented by Duff (1981) in the Harwell Subroutine Library code `MC21` has a theoretically worst-case behaviour of $\mathcal{O}(n\tau)$, but in practice it behaves more like $\mathcal{O}(n+\tau)$. Because this latter algorithm is far simpler, we concentrate on this in the following although we note that it is relatively straightforward to modify and use the algorithm of Hopcroft and Karp (1973) in a similar way.

A limitation of algorithm BT is that it only maximizes the smallest value on the diagonal of the permuted matrix. Although this means that the other diagonal values are no smaller, they may not be maximal. Consider, for example, the 3×3 matrix

$$\begin{pmatrix} \delta & 1.0 & 1.0 \\ 1.0 & \delta & \\ & & \delta \end{pmatrix} \tag{2.2}$$

with δ close to zero. Algorithm BT applied to this matrix returns $\epsilon = \delta$ and either the transversal $\{(1,1),(2,2),(3,3)\}$ or $\{(2,1),(1,2),(3,3)\}$. Clearly, the latter transversal is preferable. The modifications that we propose help to do this by choosing large entries when possible for the early transversal entries.

It is beneficial to first permute the matrix to block triangular form and then to use BT on only the blocks on the diagonal. This can be done since all entries in any maximum transversal must lie in these blocks. Furthermore, not only does this mean that BT operates on smaller matrices, but we also usually obtain a transversal of better quality inasmuch as not only is the minimum diagonal entry maximized but this is true for each block on the diagonal. Thus for matrix (2.2), the combination of an ordering to block triangular form followed by BT would yield the preferred transversal $\{(2,1),(1,2),(3,3)\}$.

There are other possibilities for improving the diagonal values of the permuted matrix which are not the smallest. One is to apply a row scaling subsequent to an initial column scaling of the matrix $\mathbf{A}$. This will increase the numerical values of all the nonzero entries in those rows for which the maximum absolute numerical value is less than one. A row scaling applied to the matrix (2.2) changes the coefficient a_{33} from δ to 1.0, and now algorithm BT will compute $\{(2,1),(1,2),(3,3)\}$ as the bottleneck transversal of the matrix (2.2). Unfortunately, such a row scaling does

not always help, as can be seen by the matrix

$$\begin{pmatrix} \delta & 1.0 & 1.0 \\ 1.0 & \delta & \\ 1.0 & & \delta \end{pmatrix}$$

with the maximum transversals

$\{(1,1),(2,2),(3,3)\}$, $\{(2,1),(1,2),(3,3)\}$, and $\{(1,3),(2,2),(3,1)\}$

all legitimate bottleneck transversals. Indeed the BT algorithm is very dependent on scaling. For example, the matrix $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ has bottleneck transversal $\{(2,1),(1,2)\}$ whereas, if it is row scaled to $\begin{pmatrix} 4 & 8 \\ 3 & 4 \end{pmatrix}$, the bottleneck transversal is $\{(1,1),(2,2)\}$.

Another possibility for improving the size of the diagonal values is to apply algorithm BT repeatedly. Without loss of generality, suppose that, after application of BT, entry a_{nn} has the smallest diagonal value. Algorithm BT can then be applied to the $(n-1) \times (n-1)$ leading principal submatrix of $\mathbf{A}$, and this could be repeated until (after k steps) the $(n-k) \times (n-k)$ leading principal submatrix of $\mathbf{A}$ only contains ones (on assumption original matrix was row and column scaled). Obviously, this can be quite expensive, since algorithm BT is applied $\mathcal{O}(n)$ times although we have a good starting point for the BT algorithm at each stage. We call this algorithm the successive bottleneck transversal algorithm. Because of this and the fact that we have found that it usually gives little improvement over BT, we do not consider it further in this paper.

2.3 Maximum Product transversals

An algorithm yielding the same transversal independent of scaling is to maximize the product of the moduli of entries on the diagonal, that is to find a permutation σ so that

$$|\prod_{i=1}^{n} a_{i\sigma_i}| \tag{2.3}$$

is maximized. This is the strategy used for pivoting in full Gaussian elimination by Olschowka and Neumaier (1996) and corresponds to obtaining a weighted bipartite matching. Olschowka and Neumaier (1996) combine a permutation and scaling strategy. The permutation, as in (2.3), maximizes the product of the diagonal entries of the permuted matrix. (Clearly the product is zero if and only if the matrix is structurally singular.) The scaling transforms the matrix into a so-called **I**-matrix, whose diagonal entries are all one and whose off-diagonal entries are all less than or equal to one.

Maximizing the product of the diagonal entries of $\mathbf{A}$ is equivalent to minimizing the sum of the diagonal entries of a matrix $\mathbf{C} = (c_{ij})$ that is defined as follows (we

here assume that $\mathbf{A} = (a_{ij})$ denotes an $n \times n$ nonnegative nonsingular matrix):

$$c_{ij} = \begin{cases} \log \overline{a}_j - \log a_{ij} & a_{ij} \neq 0 \\ 0 & a_{ij} = 0 \end{cases}$$

where $\overline{a}_j = \max_k(a_{kj})$ is the maximum absolute value in column j of matrix $\mathbf{A}$.

Minimizing the sum of the diagonal entries can be stated in terms of an assignment problem and can be solved in $\mathcal{O}(n^3)$ time for full $n \times n$ matrices or in $\mathcal{O}(n\tau \log n)$ time for sparse matrices with τ entries. A bipartite weighted matching algorithm is used to solve this problem. Applying this algorithm to $\mathbf{C}$ produces vectors u, v and a transversal T, all of length n, such that

$$\begin{cases} u_i + v_j = c_{ij} & (i,j) \in T \\ u_i + v_j \leq c_{ij} & (i,j) \notin T \end{cases}$$

If we define

$$\begin{aligned} \mathbf{D}_1 &= diag(d_1^1, d_2^1, \ldots, d_n^1), \quad d_i^1 = \exp(u_i), and \\ \mathbf{D}_2 &= diag(d_1^2, d_2^2, \ldots, d_n^2), \quad d_j^2 = \exp(v_j)/\overline{a}_j, \end{aligned}$$

then, the scaled matrix $\mathbf{B} = \mathbf{D}_1\mathbf{A}\mathbf{D}_2$ is an I-matrix. We do not do this scaling in our experiments but, unlike Olschowka and Neumaier, we use a *sparse* bipartite weighted matching whereas they only considered full matrices.

The worst case complexity of this algorithm is $\mathcal{O}(n\tau \log n)$. This is similar to BT, although in practice it sometimes requires more work than BT. We have programmed this algorithm, without the final scaling. We have called it algorithm MPD (for Maximum Product on Diagonal) and compare it with BT and `MC21` in the later sections of this paper. Note that on the matrix

$$\begin{pmatrix} 2/\epsilon & & 1 \\ & 2/\epsilon & \\ 1 & & \epsilon \end{pmatrix},$$

the MPD algorithm obtains the transversal $\{(1,1),(2,2),(3,3)\}$ whereas, for example for Gaussian elimination down the diagonal, the transversal $\{(1,3),(2,2),(3,1)\}$ would be better. Additionally, the fact that scaling does influence the choice of bottleneck transversal could be deemed a useful characteristic.

3 Implementation of the BT algorithm

We now consider implementation details of algorithm BT from the previous section. We will also illustrate its performance on some matrices from the Harwell-Boeing Collection (Duff, Grimes and Lewis 1989) and the collection of Davis (1997). A code implementing the BT algorithm will be included in a future release of the Harwell Subroutine Library (HSL 1996).

When we are updating the transversal at stage ($\star$) of algorithm BT, we can easily accelerate the algorithm described in Section 2 by computing the value of the minimum entry of the transversal, viz.

$$\min_{(i,j)\in T} |a_{ij}| \tag{3.1}$$

and then setting ϵmin to this value rather than to ϵ. The other issue, crucial for efficiency, is the choice of ϵ at the beginning of each step. If, at each step, we choose ϵ close to the value of ϵmin then it is highly likely that we will find a maximum transversal, but the total number of steps required to obtain the bottleneck transversal can be very large. In the worst case, we could require $\tau - n$ steps when the number of nonzero entries in $\mathbf{A}_\epsilon$ reduces by only one at each iteration.

The algorithm converges faster if the size of the interval $(\epsilon min, \epsilon max)$ reduces significantly at each step. It would therefore appear sensible to choose ϵ at each step so that the interval is split into two almost equal subintervals, that is $\epsilon \approx (\epsilon min + \epsilon max)/2$. However, if most of the nonzero values in $\mathbf{A}$ that have a magnitude between ϵmin and ϵmax, are clustered near one of these endpoints, the possibility exists that only a few nonzero values are discarded and the algorithm again will proceed slowly. To avoid this, ϵ should be chosen as the median of the nonzero values between ϵmin and ϵmax.

We now consider how a transversal algorithm like `MC21` can be modified to implement algorithm BT efficiently. Before doing this, it is useful to describe briefly how `MC21` works. Each column of the matrix is searched in turn (called an original column) and either an entry in a row with no transversal entry presently in the row is found and this is made a transversal entry (a *cheap assignment*) or there is no such entry and so the search moves to a previous column whose transversal entry is in one of the rows with an entry in the original column. This new column is then checked for a cheap assignment. If one exists, then this cheap assignment and the entry in the original column in the row of the old transversal entry, replace that as transversal entries thereby extending the length of the transversal by 1. If there is no cheap assignment, then the search continues to other columns in a depth first search fashion until a chain or *augmenting path* of the form

$$\{(a_{j_1,j}), (a_{j_2,j_1}), (a_{j_3,j_2}), \ldots, (a_{i,j_k})\}$$

is found where there are no transversal entries in row i and every odd member of the path is a transversal entry. The assignment is made in column j and the transversal

extended by 1 by replacing all transversal entries in the augmenting path with the even members of this path.

Transversal selection algorithms like MC21 do not take into account the numerical values of the nonzero entries. However, it is clear that the algorithm BT will converge faster if T is chosen so that the value of its minimum entry is large. We do this by noting that, when constructing an augmenting path, there are often several candidates for a cheap assignment or for extending the path. MC21 makes an arbitrary choice and we have modified it so the candidate with largest absolute value is chosen. Note that this is a local strategy and does not guarantee that augmenting paths with the highest values will be found.

The second modification aims at exploiting information obtained from previous steps of algorithm BT. Algorithm BT repeatedly computes a maximum transversal $T = MT(A_\epsilon, T'_\epsilon)$. The implementation of MC21 in the Harwell Subroutine Library computes T from scratch, so we have modified it so that it can start with a partial transversal. This can easily be achieved by holding the set of columns which contain entries of the partial transversal and performing the depth search search through that set of columns.

Of course, there are many ways to implement the choice of ϵ. One alternative is to maintain an array PTR (of length τ) of pointers, such that the entries in the first part of PTR point to those entries in $\mathbf{A}$ that form matrix $\mathbf{A}_{\epsilon_{max}}$, the first two parts of PTR point to the entries that form $\mathbf{A}_{\epsilon_{min}}$, and the elements in the third part of PTR point to all the remaining (smaller) entries of $\mathbf{A}$. A new value for ϵ can then be chosen directly ($\mathcal{O}(1)$ time) by picking the numerical value of an entry that is pointed to by an element of the second part of PTR. After the assignment in algorithm BT to either ϵ_{min} or ϵ_{max}, the second part of PTR has to be permuted so that PTR again can be divided into three parts. An alternative is to do a global sort (using a fast sorting algorithm) on all the entries of $\mathbf{A}$, such that the elements of PTR, point to the entries in order of decreasing absolute value. Then again PTR can be divided into three parts as described in the previous alternative. By choosing (in $\mathcal{O}(1)$ time) ϵ equal to the numerical value of the entry pointed to by the median element of the second part of PTR, ϵ will divide the interval $(\epsilon_{min}, \epsilon_{max})$ into parts of close-to-equal size. Both alternatives have the advantage of being able to choose the new ϵ quickly, but require $\mathcal{O}(\tau)$ extra memory and (repeated) permutations of the pointers.

We prefer an approach that is less expensive in memory and that matches our transversal algorithm better. Since MC21 always searches the columns in order, we facilitate the construction of the matrices $\mathbf{A}_\epsilon$, by first sorting the entries in each column of the matrix $\mathbf{A}$ by decreasing absolute value. For a sparse matrix with a well bounded number of entries in each column, this can be done in $\mathcal{O}(n)$ time. The matrix $\mathbf{A}_\epsilon$ is then implicitly defined by an array LEN of length n with LEN[j] pointing to the first entry in column j of matrix $\mathbf{A}$ whose value is smaller than ϵ, which is the position immediately after the end of column j of matrix $\mathbf{A}_\epsilon$. Since the

entries of a column of $\mathbf{A}_\epsilon$ are contiguous, the repeated modification of ϵ by algorithm BT, which redefines matrix $\mathbf{A}_\epsilon$, corresponds to simply changing the pointers in the array LEN.

The actual choice of ϵ at phase $(\star)$ in algorithm BT is done by selecting in matrix $\mathbf{A}_{\epsilon_{min}}$ an entry that has an absolute value X such that $\epsilon_{min} < X \leq \epsilon_{max}$. The columns of $\mathbf{A}_{\epsilon_{min}}$ are searched until such an entry is found and ϵ is set to its absolute value. This search costs $\mathcal{O}(n)$ time since, for each column, we have direct access to the entries with absolute values between ϵ_{min} and ϵ_{max} through the pointer array LEN.

As mentioned before, by choosing ϵ carefully, we can speed up algorithm BT considerably. Therefore, instead of choosing an arbitrary entry from the matrix to define ϵ, we can choose a number (k say) of entries lying between ϵ_{min} and ϵ_{max} at random, sort them by absolute value, and then set ϵ to the absolute value of the median element.[2] In our implementation we used $k = 10$.

The set of matrices that we used for our experiments are unsymmetric matrices taken from the sparse matrix collections Duff, Grimes and Lewis (1992) and Davis (1997). Table 3.1 shows the order, number of entries, and the time to compute a bottleneck transversal for each matrix. All matrices are initially row and column scaled. By this we mean that the matrix is scaled so that the maximum entry in each row and in each column is one.

The machine used for the experiments in this and the following sections is a 166 MHz SUN ULTRA-2. The algorithms are implemented in Fortran 77.

Matrix	n	τ	Time in secs		
			MC21	BT	MPD
MAHINDAS	1258	7682	0.01	0.01	0.02
ORANI678	2529	90158	0.02	0.10	0.13
RDIST1	4134	94408	0.02	0.18	0.37
GEMAT11	4929	33185	0.01	0.03	0.04
GOODWIN	7320	324784	0.27	2.26	1.82
ONETONE1	36057	341088	2.67	0.70	0.61
ONETONE2	36057	227628	2.63	0.53	0.42
TWOTONE	120750	1224224	60.10	6.95	2.17
LHR02	2954	37206	0.04	0.14	0.17
LHR14C	14270	307858	0.28	1.13	3.32
LHR71C	70304	1528092	1.86	9.00	37.73
AV41092	41092	1683902	35.72	10.81	65.13

Table 3.1: Times for transversal algorithms. Order of matrix is n and number of entries τ.

[2]This is a technique commonly used to speed up sorting algorithms like quicksort.

4 The solution of equations by direct methods

MCSPARSE, a parallel direct unsymmetric linear system solver developed by Gallivan, Marsolf and Wijshoff (1996), uses a reordering to identify a priori large and medium grain parallelism and to reorder the matrix to bordered block triangular form. Their ordering uses an initial nonsymmetric ordering that enhances the numerical properties of the factorization, and subsequent symmetric orderings are used to obtain a bordered block triangular matrix (Wijshoff 1989). The nonsymmetric ordering is effectively a modified version of MC21. During each search phase, for both a cheap assignment and an augmenting path, an entry a_{ij} is selected only if its absolute value is within a bound α, $0 \leq \alpha \leq 1$, of the largest entry in column j. Instead of taking the first entry that is found by the search that satisfies the threshold, the algorithm scans all of the column for the entry with the largest absolute value.

The algorithm starts off with an initial bound $\alpha = 0.1$. If a maximum transversal cannot be found, then the values in each column are examined to determine the maximum value of the bound that would have allowed an assignment to take place for that column. The new bound is then set to the minimum of the bound estimates from all the failed columns and the algorithm is restarted. If a bound less than a preset limit is tried and a transversal is still not found, then the bound is ignored and the code finds any transversal. In our terminology (assuming an initial column scaling of the matrix) this means that a maximum transversal of size n is computed for the matrix $\mathbf{A}_\alpha$.

In the multifrontal approach of Duff and Reid (1983), later developed by Amestoy and Duff (1989), an analysis is performed on the structure of $\mathbf{A} + \mathbf{A}^T$ to obtain an ordering that reduces fill-in under the assumption that all diagonal entries will be numerically suitable for pivoting. The numerical factorization is guided by an assembly tree. At each node of the tree, some steps of Gaussian elimination are performed on a dense submatrix whose Schur complement is then passed to the parent node in the tree where it is assembled (or summed) with Schur complements from the other children and original entries of the matrix. If, however, numerical considerations prevent us from choosing a pivot then the algorithm can proceed, but now the Schur complement that is passed to the parent is larger and usually more work and storage will be needed to effect the factorization.

The logic of first permuting the matrix so that there are large entries on the diagonal, before computing the ordering to reduce fill-in, is to try and reduce the number of pivots that are delayed in this way thereby reducing storage and work for the factorization. We show the effect of this in Table 4.1 where we can see that even using MC21 can be very beneficial although the BT algorithm can show significant further gains and sometimes the use of MPD can cause further significant reduction in the number of delayed pivots. We should add that the numerical accuracy of

the solution is sometimes slightly improved by these permutations and, in all cases, good solutions were found.

Matrix	Transversal algorithm used			
	None	MC21	BT	MPD
GEMAT11	-	76	0	0
ONETONE1	-	16261	255	100
ONETONE2	40916	8310	214	100
GOODWIN	536	1622	358	53
LHR02	3432	388	211	56
LHR14C	-	7608	3689	169
LHR71C	-	35354	-	2643
AV41092	-	10151	2143	1730

Table 4.1: Number of delayed pivots in factorization from MA41. An "-" indicates that MA41 requires a real working space larger than 25 million words (of 8 bytes).

In Table 4.2, we show the effect of this on the number of entries in the factors. Clearly this mirrors the results in Table 4.1 and shows the benefits of the transversal selection algorithms. This effect is seen in Table 4.3 where we can sometimes observe a dramatic reduction in time for solution when preceded by a permutation.

Matrix	Transversal algorithm used			
	None	MC21	BT	MPD
GEMAT11	-	127819	78589	78161
ONETONE1	-	10359161	6957229	4715085
ONETONE2	14082683	2875603	2167523	2169903
GOODWIN	1263104	2673318	1791112	1282004
LHR02	2298550	333450	394724	235048
LHR14C	-	3111142	4596028	2164392
LHR71C	-	18786982	-	11599556
AV41092	-	16226036	15140098	14110336

Table 4.2: Number of entries in the factors from MA41.

In addition to being able to select the pivots chosen by the analysis phase, the multifrontal code MA41 will do better on matrices whose structure is symmetric or nearly so. The transversal orderings in some cases increase the symmetry of the resulting reordered matrix. This is particularly apparent when we have a very sparse system with many zeros on the diagonal. In that case, the reduction in number of off-diagonal entries in the reordered matrix has an influence on the symmetry. Notice that, in this respect, the more sophisticated transversal algorithms may actually cause problems since they could reorder a symmetrically structured matrix with a zero-free diagonal, whereas MC21 will leave it unchanged.

Matrix	Transversal algorithm used			
	None	MC21	BT	MPD
GEMAT11	-	0.28	0.20	0.20
ONETONE1	-	225.71	83.11	44.22
ONETONE2	81.45	17.05	9.48	11.54
GOODWIN	3.64	14.63	6.00	3.56
LHR02	24.85	1.07	1.29	0.58
LHR14C	-	12.66	29.17	5.88
LHR71C	-	148.07	-	43.33
AV41092	-	226.20	184.68	155.70

Table 4.3: Time (in seconds on Sun ULTRA-2) for MA41 for solution of system.

5 The solution of equations by iterative methods

A large family of iterative methods, the so-called stationary methods, has the iteration scheme

$$\mathbf{M}\mathbf{x}^{(k+1)} = \mathbf{N}\mathbf{x}^{(k)} + \mathbf{b}, \tag{5.1}$$

where $\mathbf{A} = \mathbf{M} - \mathbf{N}$ is a *splitting* of $\mathbf{A}$, and $\mathbf{M}$ is chosen such that a system of the form $\mathbf{M}\mathbf{x} = \mathbf{y}$ is easy to solve. If $\mathbf{M}$ is invertible, (5.1) can be written as

$$\mathbf{x}^{(k+1)} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b} = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b}\ . \tag{5.2}$$

We have

$$\rho(\mathbf{M}^{-1}\mathbf{N}) \leq ||\mathbf{M}^{-1}\mathbf{N}||_\infty \leq ||\mathbf{M}^{-1}||_\infty ||\mathbf{N}||_\infty,$$

where ρ is the spectral radius, so that, if $||\mathbf{M}^{-1}||_\infty ||\mathbf{N}||_\infty < 1$, convergence of the iterates $\mathbf{x}^{(k)}$ to the solution $\mathbf{A}^{-1}\mathbf{b}$ is guaranteed for arbitrary $\mathbf{x}^{(0)}$. In general, the smaller $||\mathbf{M}^{-1}||_\infty ||\mathbf{N}||_\infty$, the faster the convergence. Thus an algorithm that makes entries in $\mathbf{M}$ large and those in $\mathbf{N}$ small should be beneficial.

The most simple method of this type is the Jacobi method, corresponding to the splitting $\mathbf{M} = \mathbf{D}$ and $\mathbf{N} = -(\mathbf{L} + \mathbf{U})$, where $\mathbf{D}$ denotes the diagonal, $\mathbf{L}$ the strictly lower triangular part, and $\mathbf{U}$ the strictly upper triangular part of the matrix $\mathbf{A}$. However, this is not a particularly current or powerful method so we conduct our experiments using the block Cimmino implementation of Arioli, Duff, Noailles and Ruiz (1992), which is equivalent to using a block Jacobi algorithm on the normal equations. In this implementation, the subproblems corresponding to blocks of rows from the matrix are solved by a direct method similar to that considered in the previous section. For similar reasons, it can be beneficial to increase the magnitude of the diagonal entries through unsymmetric permutations.

We show the effect of this in Table 5.1, where we see that the number of iterations for the solution of the problem MAHINDAS ($n = 1258$, $\tau = 7682$). The convergence tolerance was set to 10^{-12}. The transversal selection algorithm was followed by a reverse Cuthill McKee algorithm to obtain a block tridiagonal form. The matrix

was partitioned in 2, 4, 8, and 16 block rows and the acceleration used was a block CG algorithm with block sizes of 1, 4, and 8.

Acceleration + # block rows	Transversal algorithm None	MC21	BT	MPD
CG(1)				
2	324	267	298	295
4	489	383	438	438
8	622	485	532	524
16	660	572	574	574
CG(4)				
2	148	112	130	133
4	212	190	199	194
8	261	235	232	233
16	281	245	253	253
CG(8)				
2	80	62	72	75
4	117	105	109	108
8	140	133	127	130
16	151	142	137	136

Table 5.1: Number of iterations of block Cimmino algorithm on MAHINDAS.

In every case, the use of a transversal algorithm accelerates the convergence of the method, sometimes by a significant amount. However, the use of the algorithms to increase the size of the diagonal entries does not usually help convergence further. The convergence of block Cimmino depends on angles between subspaces which is not so strongly influenced by the diagonal entries.

6 Preconditioning

In this section, we consider the effect of using a permutation induced by our transversal algorithms prior to solving a system using a preconditioned iterative method. We consider preconditionings corresponding to incomplete factorizations of the form ILU(0), ILU(1), and ILUT and study the convergence of the iterative methods GMRES(20), BiCGSTAB, and QMR. We refer the reader to a standard text like that of Saad (1996) for a description and discussion of these methods. Since the diagonal of the permuted matrix is "more dominant" than the diagonal of the original matrix, we would hope that such permutations would enhance convergence.

We show the results of some of our runs in Table 6.1. The maximum number of iterations was set to 1000 and the convergence tolerance to 10^{-9}. It is quite clear that the reorderings can have a significant effect on the convergence of the

preconditioned iterative method. In some cases, the method will only converge after the permutation, in others it greatly improves the convergence. It would appear from the results in Table 6.1 and other experiments that we have performed, that the more sophisticated MPD transversal algorithm generally results in the greatest reduction in the number of iterations, although the best method will depend on the overall solution time including the transversal selection algorithm.

7 Conclusions and future work

We have described algorithms for obtaining transversals with large entries and have indicated how they can be implemented showing that resulting programmes can be written for efficient performance.

While it is clear that reordering matrices so that the permuted matrix has a large diagonal can have a very significant effect on solving sparse systems by a wide range of techniques, it is somewhat less clear that there is a universal strategy that is best in all cases. We have thus started experimenting with combining the strategies mentioned in this paper and, particularly for the block Cimmino approach, with combining our unsymmetric ordering with a symmetric ordering. One example that we plan to study is a combination with the symmetric TPABLO ordering (Benzi, Choi and Szyld 1997).

It is possible to extend our techniques to orderings that try to increase the size of not just the diagonal but also the immediate sub and super diagonals and then use the resulting tridiagonal part of the matrix as a preconditioner.

One can also build other criteria into the weighting for obtaining a bipartite matching, for example, to incorporate a Markowitz count so that sparsity would also be preserved by the choice of the resulting diagonal as a pivot.

Finally, we noticed in our experiments with `MA41` that one effect of transversal selection was to increase the structural symmetry of unsymmetric matrices. We are thus exploring further the use of ordering techniques that more directly attempt to increase structural symmetry.

Acknowledgments

We are grateful to Patrick Amestoy of ENSEEIHT, Michele Benzi of CERFACS, and Daniel Ruiz of ENSEEIHT for their assistance with the experiments on the direct methods, the preconditioned iterative methods, and the block iterative methods respectively. We would also like to thank Alex Pothen for some early discussions on bottleneck transversals, and John Reid and Jennifer Scott for comments on a draft of this paper.

Matrix and method		Transversal algorithm		
		MC21	BT	MPD
IMPCOL E				
ILU(0)	GMRES(20)	-	16	15
	BiCGSTAB	123	21	11
	QMR	101	26	17
ILU(1)	GMRES(20)	59	15	11
	BiCGSTAB	98	16	8
	QMR	72	19	12
ILUT	GMRES(20)	8	7	8
	BiCGSTAB	9	5	5
	QMR	10	8	8
MAHINDAS				
ILU(0)	GMRES(20)	-	-	179
	BiCGSTAB	-	-	39
	QMR	-	-	55
ILU(1)	GMRES(20)	-	-	69
	BiCGSTAB	-	-	26
	QMR	-	-	34
ILUT	GMRES(20)	-	-	15
	BiCGSTAB	-	-	11
	QMR	-	-	17
WEST0497				
ILU(0)	GMRES(20)	-	60	19
	BiCGSTAB	-	78	22
	QMR	-	63	23
ILU(1)	GMRES(20)	-	79	15
	BiCGSTAB	-	82	15
	QMR	-	82	18
ILUT	GMRES(20)	-	15	10
	BiCGSTAB	-	15	7
	QMR	-	17	12

Table 6.1: Number of iterations required by some preconditioned iterative methods.

References

Amestoy, P. R. and Duff, I. S. (1989), 'Vectorization of a multiprocessor multifrontal code', *Int. J. of Supercomputer Applics.* **3**, 41–59.

Arioli, M., Duff, I. S., Noailles, J. and Ruiz, D. (1992), 'A block projection method for sparse matrices', *SIAM J. Scientific and Statistical Computing* **13**, 47–70.

Benzi, M., Choi, H. and Szyld, D. (1997), Threshold ordering for preconditioning nonsymmetric problems, Technical Report TR/PA/97/02, CERFACS, Toulouse, France. (Submitted to Hong Kong Workshop on Scientific Computing 1997).

Davis, T. A. (1997), 'University of Florida sparse matrix collection', Available at http://www.cise.ufl.edu/~davis and ftp://ftp.cise.ufl.edu/pub/faculty/davis.

Duff, I. S. (1977), MA28 – A set of Fortran subroutines for sparse unsymmetric linear equations, Technical Report AERE R8730, Her Majesty's Stationery Office, London.

Duff, I. S. (1981), The design and use of a frontal scheme for solving sparse unsymmetric equations, *in* J. P. Hennart, ed., 'Numerical Analysis, Proceedings of 3rd IIMAS Workshop. Lecture Notes in Mathematics 909', Springer Verlag, Berlin, pp. 240–247.

Duff, I. S. and Reid, J. K. (1983), 'The multifrontal solution of indefinite sparse symmetric linear systems', *ACM Trans. Math. Softw.* **9**, 302–325.

Duff, I. S. and Wiberg, T. (1988), 'Remarks on implementation of $\mathcal{O}(n^{1/2}\tau)$ assignment algorithms', *ACM Trans. Math. Softw.* **14**(3), 267–287.

Duff, I. S., Grimes, R. G. and Lewis, J. G. (1989), 'Sparse matrix test problems', *ACM Trans. Math. Softw.* **15**(1), 1–14.

Duff, I. S., Grimes, R. G. and Lewis, J. G. (1992), Users' guide for the Harwell-Boeing sparse matrix collection (Release I), Technical Report RAL 92-086, Rutherford Appleton Laboratory.

Fulkerson, D., Glicksberg, I. and Gross, O. (1953), A production line assignment problem, Technical Report RM-1102, RAND Corporation.

Gallivan, K. A., Marsolf, B. A. and Wijshoff, H. A. G. (1996), 'Solving large nonsymmetric sparse linear systems using MCSPARSE', *Parallel Computing* **22**, 1291–1333.

Hopcroft, J. E. and Karp, R. M. (1973), 'An $n^{(5/2)}$ algorithm for maximum matchings in bipartite graphs', *SIAM J. Comput.* **2**, 225–231.

HSL (1996), *Harwell Subroutine Library. A Catalogue of Subroutines (Release 12)*, AEA Technology, Harwell Laboratory, Oxfordshire, England. For information concerning HSL contact: Dr Scott Roberts, AEA Technology, 552 Harwell, Didcot, Oxon OX11 0RA, England (tel: +44-1235-434988, fax: +44-1235-434136, email: Scott.Roberts@aeat.co.uk).

Olschowka, M. and Neumaier, A. (1996), 'A new pivoting strategy for Gaussian elimination', *Linear Algebra and its Applications* **240**, 131–151.

Pothen, A. and Fan, C. (1990), 'Computing the block triangular form of a sparse matrix', *ACM Trans. Math. Softw.* **16**(4), 303–324.

Saad, Y. (1996), *Iterative methods for sparse linear systems*, PWS Publishing, New York, NY.

Wijshoff, H. A. G. (1989), Symmetric orderings for unsymmetric sparse matrices, Technical Report CSRD 901, Center for Supercomputing Research and Development, University of Illinois.