Ordering Algorithms for Irreducible Sparse Linear Systems

by

R. Fletcher* and J.A.J.Hall**

Abstract

Ordering algorithms aim to pre-order a matrix in order to achieve a favourable structure for factorization. Two prototype structures are described which are commonly aimed at by current algorithms. A different structure based on a lower Hessenberg form is introduced. We show that the common structures may be obtained from the Hessenberg form in a simple way. A fundamental and very simple algorithm is presented for deriving the lower Hessenberg form. This algorithm is inherently a part of other common algorithms, but is usually obscured by other detailed heuristics. Some of these heuristics are seen to correspond to different tie-break rules in the fundamental algorithm. We describe a particularly simple tie-break rule used in SPK1 [11], which is effective and not at all well known.

Ordering algorithms need to be modified to enable pivoting for numerical stability to be carried out. We describe how the idea of threshold pivoting can be used in the context of these algorithms. Most ordering algorithms in common use employ some sort of look-ahead to promote a favourable structure. It is argued that look-ahead is generally ineffective in practice, and that numerical evidence supports the use of very simple strategies.

A new ordering algorithm is presented which aims to obtain a narrower bandwidth in the lower Hessenberg form. Some limited numerical experience is presented which enables us to draw tentative conclusions about various ordering strategies, and how they compare with those in common use.

1. Introduction

Many techniques have been suggested for the solution of a large sparse nonsingular system of linear equations $Ax = b$, and related problems. The most effective method in any particular case is very much dependent on context, and this paper relates to one particular approach which has been used in a number of applications. We start by assuming that A has been reduced to block lower triangular form. This can be achieved first by finding a transversal using an algorithm such as that of Duff [3], followed by the use of Tarjan's algorithm ([13],[5]), based on the use of a symmetric row and column permutation. Each diagonal block in the resulting matrix is then irreducible and it is sufficient to find LU factors of each block in order to solve $Ax = b$ efficiently (see for example [6]). There is often considerable sparsity within the diagonal blocks, and it is worthwhile to continue to use sparse matrix techniques to obtain a favourable structure

* Dept. of Mathematics and Computer Science, Univ. of Dundee
** Dept. of Mathematics, Univ. of Edinburgh

in these LU factors. It is the problem of ordering and factorizing matrices such as these diagonal blocks that we address in this paper. Subsequently therefore, A refers to an irreducible nonsingular matrix that cannot be further simplified by Tarjan's algorithm.

A popular way to calculate LU factors of A is to use the Markowitz pivot strategy ([10],[6]). This chooses a pivot which minimizes $(r_i - 1)(c_j - 1)$ subject to a threshold pivot tolerance, where r_i and c_j are the row and column counts in the reduced matrix in Gaussian elimination. It has been observed that this method is effective for keeping fill-in low in the LU factors. However the need to make provision for fill-in, and to have row and column counts for the reduced matrix at every stage, means that a complex data structure must be adopted during the factorization, and this can increase the run time.

There is therefore some interest in alternatives to the Markowitz approach, and a common theme in a number of algorithms is to pre-order the matrix A in order to achieve a favourable structure for factorization. Such methods are the subject of this paper. In Section 2 we describe two prototype structures which are commonly aimed at by current algorithms. We also argue the merits of introducing a different structure based on a lower Hessenberg form. We show that the common structures may be obtained from the Hessenberg form in a simple way. A fundamental and very simple algorithm is presented for deriving the lower Hessenberg form. This algorithm is inherently a part of most other algorithms, but is usually obscured by other detailed heuristics, and we feel that it is valuable to make its framework clear. Some of these heuristics are seen to correspond to different tie-break rules in the fundamental algorithm. Other heuristics determine which of the common structures is obtained. We describe a particularly simple tie-break rule, used in SPK1 [11], which is effective and not at all well known.

Most ordering algorithms in common use employ some sort of look-ahead to promote a favourable structure. Two different possibilities are described in Section 3. We argue that look-ahead is generally ineffective in practice, and that numerical evidence supports the use of very simple strategies.

Ordering algorithms need to be modified to enable pivoting for numerical stability to be carried out. We show how the idea of threshold pivoting can be used in the context of these algorithms. There are two main aspects of interest, one relating to the use of a bordered block lower triangular form, and the other to sparse Gaussian elimination on a lower triangular form with a number of columns projecting above the diagonal. Both of these situations are discussed in Section 4.

In Section 5 we present a new ordering algorithm aimed at obtaining a narrower bandwidth in the lower Hessenberg form. This has some intrinsic interest, although our numerical experience is not very encouraging. However the development does indicate that there are many potential strategies for ordering algorithms which have not yet been researched, and points to how little is known about the important features of such algorithms. Some limited numerical experience is presented in Section 6 which enables us to draw tentative conclusions about various ordering strategies, and how they compare with those in common use.

2. Ordering Algorithms

The basic ideas of ordering algorithms are introduced by Hellerman and Rarick [9],

who suggest two algorithms known as P^3 and P^4. P^4 is a development of P^3 which allows for the prior reduction of A by Tarjan's algorithm, and we need only consider P^4. A later paper by Erisman *et al.* [7] introduces a modification to P^4 and a further modification known as P^5. The aim of these algorithms is essentially to make both row and column permutations of A so that the resulting matrix is close in some sense to a lower or block lower triangular matrix.

The characteristic structure produced by the P^4 algorithm—as originally conceived by Hellerman and Rarick—is that of a lower triangular matrix with a number of columns projecting above the diagonal. Such columns are termed *spikes* and we refer to this structure as being *spiked lower triangular*. The spike columns are accumulated as the algorithm proceeds in a *spike stack*. During the algorithm, spikes are removed from the stack, to be included as columns in the reordered matrix. Upon termination of the P^4 algorithm this spike stack is empty. Figures 2.1(a) and 2.1(b) provide an illustration of this ordering on a matrix of dimension 44.

The spiked lower triangular form is most useful when the number of spikes is relatively small (assuming that an appropriate data structure is used). The simplest way to use the structure is to apply Gaussian elimination without pivoting, in which case it is readily observed that fill-in only takes place within the spike columns. It may be however that a diagonal element of the ordered matrix is *structurally zero* (see [6]) in which case this process fails. It is also possible that near-zero diagonal pivots can occur which give rise to numerical growth in the spikes, and hence to numerical instability. Hence any practical implementation of this approach must allow for pivotal interchanges, and we consider this aspect in more detail in Section 4.

The P^5 algorithm produces a related structure in which the reordered matrix is in *bordered block lower triangular* form, that is

$$PAQ = \begin{bmatrix} L & B \\ C & D \end{bmatrix} \tag{2.1}$$

where L is a block lower triangular matrix whose diagonal blocks are square and fully dense, and P and Q are permutation matrices. The P^5 algorithm attains this form by restricting the removal of columns from the spike stack, whilst also accumulating certain rows in a row stack. On termination of the algorithm, the spike stack is not empty and is equal in size to the row stack. The rows and columns that remain in these stacks form the border of the resulting bordered block lower triangular matrix. A typical profile resulting from this ordering is illustrated in Figure 2.1(c).

The modification to the P^4 algorithm introduced by Erisman *et al.* [6] and implemented by Arioli, *et al.* [2] also obtains a bordered block triangular matrix. The diagonal blocks are larger than those obtained by the P^5 algorithm and are not fully dense, indeed they may require to be treated as sparse matrices. The compensation for this considerable increase in complexity is a marginally smaller border.

The form of (2.1) ensures that the diagonal blocks of L are structurally nonsingular, and so avoids the difficulties associated with structurally zero pivots referred to above. Unfortunately it is possible that *numerically* zero or near-zero pivots in L may occur, and this approach must be supplemented by a pivotal strategy. We return to this point in Section 4.

(a) Unordered matrix *(b) Spiked lower triangular*

(c) Bordered block lower triangular *(d) Lower Hessenberg*

Figure 2.1 Alternative forms for ordered matrices

The structure of (2.1) is utilised by taking advantage of the ease of solving systems involving L. To do this only the diagonal blocks of L need be factorized. In practice it is usually observed that these blocks are trivial (1×1 and occasionally 2×2). Blocks of higher order occur rarely and the 4×4 block in Figure 2.1(c) is remarkable. Although this matrix is small, the residual effect of block structure in a large sparse problem could result in similar behaviour. This implicit factorization of L is used to form the *Schur complement*

$$S = D - CL^{-1}B \tag{2.2}$$

Next LU factors of the matrix S (with pivoting) are calculated using dense code since S is typically very dense, if not full. Systems involving A can then be solved by a

combination of solves with L and S. It can be advantageous to store the matrix $L^{-1}B$ (or CL^{-1}), so that a solve with A may be carried out by using only one additional solve with L. This approach is most efficient when the border (and hence the Schur complement) is small.

The P^4 and P^5 algorithms are examples of one-pass ordering algorithms in that a row or column is assigned to a new position exactly once. If the twin processes of identifying and assigning rows and columns within the one pass are viewed as separate processes, each within a single pass of an equivalent two-pass process, then considerable insight into the operation of these and other algorithms is obtained.

There is also another characteristic form associated with these algorithms which we refer to as the *lower Hessenberg form*. If the spike columns are included with the other columns of A so that the height of the columns is non-increasing then a lower Hessenberg matrix is obtained. This matrix has the particular property that it is block lower triangular, but with fully dense rectangular blocks on the diagonal, rather than square blocks. Since the matrix is irreducible the profile always lies above the diagonal. This structure is illustrated in Figure 2.1(d).

We give here a fundamental and very simple algorithm for determining a lower Hessenberg form of this nature. The algorithm progressively removes rows and columns from A: the matrix that remains when some rows and columns have been removed is termed the *active submatrix*. A count is kept of the numbers of nonzero elements in the rows and columns of the active submatrix. Initially A is the active submatrix and the algorithm can be simply expressed as follows.

repeat
1) find a row in the active submatrix with minimum row count
2) remove all columns which intersect this row
3) update row counts in the active submatrix
4) remove all rows with zero row count
until all rows and columns are removed. (2.3)

In this algorithm columns are removed by permuting them to the left of the active submatrix, and rows by permuting them to the top of the active submatrix. At each stage the block that is common to the rows and columns being removed forms a rectangular diagonal block of the lower Hessenberg matrix. The selection of a minimum-row-count row in 1), and the removal of zero-row-count rows in 4), ensures that the rectangular diagonal blocks are fully dense. A typical intermediate stage of algorithm (2.3) is illustrated in Figure 2.2.

Various algorithms exist which are essentially based on this common framework, and they differ in how they break ties in step 1). We discuss aspects of this in the rest of this section. However it is worth pointing out that the row that is selected in step 1) uniquely determines the rectangular block that is obtained. Hence any tie-break rule essentially selects between the different candidate blocks that can be formed from minimum-row-count rows. It follows for example that if there is only one candidate block, then there is no need for a tie-break, even though the minimum-row-count row is not uniquely defined.

The P^5 algorithm and both interpretations of the P^4 algorithm are based on the same rather complicated way of breaking ties and hence correspond to the same lower

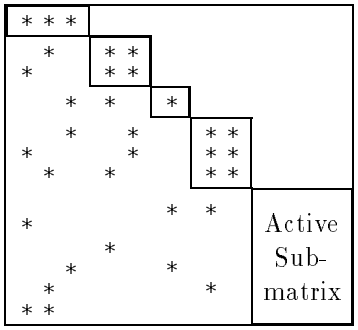

Figure 2.2 Intermediate stage for Algorithm (2.3)

Hessenberg matrix. We refer to this ordering as the *HR ordering* since it is characteristic of the Hellerman-Rarick family of ordering algorithms. We give more details of this means of breaking ties in Section 3. However there is a different and much more simple tie break rule which is to *select a row which maximizes the number of nonzero entries in the columns being removed in step 2).* This needs the column counts for the active submatrix, which are the same as the column counts in the original matrix A. It is possible that ties may still remain when this rule is used, in which case they can be resolved in an arbitrary way.

We originally believed this tie break rule to be new, but found that it was implicitly contained within the SPK1 ordering algorithm of Stadtherr and Wood [11], which obtains a spiked lower triangular matrix. Because of this we refer to the the ordering as the *SW ordering* when it is used to determine a lower Hessenberg matrix. The SPK1 algorithm seems not to be well known, except possibly amongst the chemical engineering fraternity, and we are grateful to Prof. A. W. Westerberg for bringing the papers to our attention. The main advantage of the SPK1 algorithm is that it is much cheaper to calculate the ordering than with the P^4 and P^5 methods. It is of some interest to know how the resulting orderings compare, and in particular whether the number of spikes, or the dimension of the border, is greater or less. Practical experience, both here and in [12], indicates that although there are specific examples that favour either approach, there is no uniform bias in any one direction. As a consequence the SPK1 algorithm is to be preferred because it is simple to code, and cheap to calculate.

In describing algorithms in terms of the lower Hessenberg form, we are implicitly suggesting a two-pass approach in which such a lower Hessenberg matrix is obtained during the first pass. In the second pass the rows and columns are assigned to final positions prior to factorization, which may involve changing to a different form, and will include considerations of pivoting. However in practice it may be possible to implement the whole process in a single pass.

To obtain a spiked lower triangular matrix from the Hessenberg form is particularly simple, requiring only a column permutation, and is described in the following rule, initialized by $k = 0$.

repeat
  let the current rectangular diagonal block contain m rows and n columns
  if $m \leq n$ then
    assign m columns of the block to locations $k+1, \ldots, k+m$ and put the remaining $n-m$ columns onto the spike stack
  else
    assign the n columns of the block and the $m-n$ columns from the top of the spike stack to locations $k+1, \ldots, k+m$
  endif
  set $k = k+m$
until all columns have been assigned.

This process is used by both the P^4 and SPK1 algorithms, which transfer the first m columns of a block to the spike stack when $m \leq n$. A bordered block triangular matrix may similarly be obtained from the lower Hessenberg form in the following way.

repeat
  let the current rectangular diagonal block contain m rows and n columns
  if $m \leq n$ then
    assign the m rows and m of the columns of the block to row/column locations $k+1, \ldots, k+m$ and put the remaining $n-m$ columns onto the spike stack
  else
    assign n of the rows and the n columns of the block to row/column locations $k+1, \ldots, k+m$ and put the remaining $m-n$ rows onto the row stack
  endif
  $k = k+m$
until all rows and columns have been assigned or placed on a stack
form the border from the row stack and spike stack.

The P^5 ordering may be obtained from the HR ordering in this way, again by transferring the first m columns of a block to the spike stack when $m \leq n$.

3. Look-Ahead

Some ordering algorithms include a feature that ties in step 1) of algorithm (2.3) are broken by looking ahead in an attempt to maximize the number of rows that are removed at the next iteration of algorithm (2.3). This would have the effect of keeping the profile of the lower Hessenberg form as close to the diagonal as possible. This usually increases the ordering time significantly, but it is hoped that this is balanced by obtaining a superior ordering.

The P^4 and P^5 algorithms both use the same look-ahead feature, which is the following. When a tie occurs in step 1) a set of candidate columns (those which intersect the tied rows) is determined. The column in this set which intersects the maximum

number of tied rows is removed and placed on the stack, and the row counts of the active submatrix are updated. This is done with the aim of reducing the row-count of as many of the tied rows as possible. In most cases ties amongst columns of maximal intersection are broken by selecting the column of maximum column count. The above process is repeated until the count of the (subset of the original) tied rows is one. At this point ties amongst columns of maximal intersection are broken by selecting the column which reduces the row-count of as many rows of next largest row-count as possible (since these rows will be the tied rows in the next iteration). Clearly at least one of the original tied rows will have had its row count reduced to zero and so can be identified as the row selected in step 1). This algorithm is therefore a realization of algorithm (2.3) in which the removal of columns and updating of the row counts in steps 2) and 3) is performed naturally as a result of the tie-breaking process in step 1).

Another look-ahead algorithm is given by Statdherr and Wood [11] which they refer to as SPK2. When a tie occurs in step 1) of (2.3), the first option is to consider, for each tied block. the number of minimum-row-count rows that would arise at the next iteration of (2.3). The tie is then broken in such a way as to maximize this number. Any further ties which remain are then broken in the same way as for SPK1.

Unfortunately, practical experience in Section 6 and in [12] gives no indication that the use of look-ahead in P^4 or SPK2 gives a superior structure to that obtained by SPK1. Numerical evidence is very problem-dependent, but there is no uniform trend which favours any particular algorithm. In view of this we are led to favour an algorithm such as SPK1 which avoids the expense of the look-ahead calculation.

4. Pivoting

Unfortunately the ordering algorithms discussed in the previous section cannot be used without some modification because of considerations of numerical stability. Various papers (for example [1],[2],[4],[12]) have addressed this problem, and it is undeniable that adequately stable factors are only attained at the expense of a loss of simplicity in the algorithm and possibly the data structure. This discussion here is carried out first of all in the context of the bordered form (2.1), although similar considerations relate to the spiked lower triangular matrix form.

The most obvious difficulty with (2.1) is that one of the diagonal blocks of L could be singular (for example the block might be of the form $\left[\begin{smallmatrix}4 & 2\\ 2 & 1\end{smallmatrix}\right]$), even though A itself is nonsingular. Then the Schur complement S in (2.2) would not exist and the algorithm would break down. Equally serious is the possibility that a diagonal block in L is nearly singular, thus causing significant growth in S, and hence significant round-off errors. However it is possible to modify any algorithm represented by (2.3) to alleviate these difficulties to a large extent. These modifications can be incorporated into the logic of (2.3) so that only one pass through the matrix is required to determine L.

The modified algorithm has the following features.

(a) Each rectangular block in the lower Hessenberg matrix is factorized using any convenient pivot strategy (for example, even complete pivoting can be used as the blocks are usually small.)

(b) Any rows and columns *not* chosen as pivot rows or columns are moved into the border.

(c) If a near-zero pivot is detected, the corresponding row and column is moved into the border.

Implementation of item (c) requires a test for a near-zero pivot: most suitable would be to detect the possibility of serious growth in the matrix S.

The most unfavourable situation for such an algorithm would be typified by a matrix such as

$$A = \begin{bmatrix} \mathbf{10} & \mathbf{1} & & & & & & \\ 1 & 10 & \mathbf{1} & & & & & \\ & 1 & 10 & \mathbf{1} & & & & \\ & & 1 & 10 & \mathbf{1} & & & \\ & & & 1 & 10 & \mathbf{1} & & \\ & & & & 1 & 10 & \mathbf{1} & \\ & & & & & 1 & 10 & \mathbf{1} \\ & & & & & & 1 & \mathbf{10} \end{bmatrix}. \tag{4.1}$$

The lower Hessenberg form obtained from this matrix is A itself, and elements in rectangular blocks are indicated in bold. Assuming that the unit elements are not regarded as being near-zero for the purposes of (c), the bordered block triangular form that is obtained is

$$PAQ = \begin{bmatrix} 10 & & & & & & & 1 \\ 1 & 1 & & & & & & 10 \\ & 10 & 1 & & & & & 1 \\ & 1 & 10 & 1 & & & & \\ & & 1 & 10 & 1 & & & \\ & & & & & 1 & 10 & \\ & & & & 1 & 10 & 1 & \end{bmatrix}. \tag{4.2}$$

The Schur complement for an $n \times n$ matrix of this form exhibits exponential growth and is $\sim 10^{n-2}$. However, whilst this is likely to be catastrophic in practice, it is typical of what can occur with any method based on threshold pivoting, with a maximum growth factor of ~ 10 per step. For example the matrix

$$A = \begin{bmatrix} 1 & & & & & & 1 \\ -10 & 1 & & & & & \epsilon \\ & -10 & 1 & & & & \epsilon \\ & & -10 & 1 & & & \epsilon \\ & & & \ddots & \ddots & & \vdots \\ & & & \epsilon & -10 & \epsilon \end{bmatrix} \tag{4.3}$$

is similarly unfavourable for the Markowitz method with threshold row pivoting. It may therefore be that researchers are prepared to accept these situations, much as they usually accept the possibility of exponential growth associated with Gaussian elimination with row pivoting.

An alternative approach to factorizing the preordered matrix in a stable way is simply to carry out Gaussian elimination with full column pivoting on the lower Hessenberg

form. That is to say, pivots may be chosen from anywhere in the lower Hessenberg profile, rather than being restricted to the blocks of L. The simplest possibility is to use a dense data structure within the lower Hessenberg profile. This requires between $\frac{1}{2}n^2$ and n^2 storage, and the flop count is $O(bn^2)$ where b is the upper bandwidth. This approach is particularly attractive if vectorized code can be used. The column operations using long vectors will speed up well on a vector machine (until the active submatrix of the elimination becomes small).

Another approach which takes more account of sparsity is to store columns of A in sparse format, allowing for fill-in, and to carry out Gaussian elimination on the spiked lower triangular form, with full threshold column pivoting. Two possible strategies suggest themselves. One chooses the pivot closest to the diagonal and so aims to preserve the *a priori* ordering as much as possible. An elimination scheme based on this criterion is referred to as LU1SOL in [12]. The second strategy is to choose the pivot to minimize the number of entries in the pivot column. This is essentially the approach used by Duff and Reid [4] and we refer to it as MINCOL. This strategy has the same motivation as Markowitz, but here the row minimization is pre-determined by the ordering method. Practical experience (Section 6) indicates that the fill-in is worse than that obtained by Markowitz, but this is balanced by the more simple data structure. It is desirable to switch to dense code when the density of the reduced matrix is sufficiently high, and this has to be taken into account when comparing fill-in. This discussion also suggests that the SPK1 ordering might not be best for these purposes since the tie break rule is to *maximize* the number of elements in the columns being removed, which is not in the Markowitz style. Rather it might be preferable to break ties by minimizing the number of elements that are removed. This would lead to more nonzeros in the active submatrix and hence the likelihood of a wider band in the bottom corner of the lower Hessenberg form. This might not be so important however, particularly if the algorithm has the ability to switch to dense code at this stage. We investigate this possibility in Section 6, with interesting results.

5. A Two-Sided Ordering

The simple form of algorithm (2.3) leads us to suggest another ordering algorithm in which *either* a minimum-row-count row *or* a minimum-column-count column is removed from the active submatrix at each iteration. The aim of this modification would be to produce a lower Hessenberg form having a smaller bandwidth. The algorithm is initialized as for (2.3) and proceeds as follows.

repeat
1) find either a minimum-row-count row or a minimum-column-count column, whichever has fewer entries
2) if a row is chosen, proceed as in algorithm (2.3)
3) if a column is chosen then (5.1)
   3a) remove all rows which intersect this column
   3b) update column counts
   3c) remove all columns with zero column count
until all rows and columns are removed.

It is suggested that ties in step 1) are broken in a similar way to SPK1, that is by maximizing the number of nonzero entries that are removed from the active submatrix.

After a column is chosen in step 3), columns are removed to the right of the active submatrix and rows are removed to below the active submatrix. Thus the overall effect of the algorithm is to generate the lower Hessenberg profile by working from both ends, as indicated in Figure 5.1. It is observed that algorithm (2.3) often gives rise to a large bandwidth near the bottom right hand corner of the ordered matrix. This algorithm would be expected to give the largest bandwidth near the centre of the matrix, at the point at which the active submatrix becomes empty. It had been hoped that the overall bandwidth would also be smaller, although practical experience does not support this. We refer to this ordering as the *Two-sided ordering* and the corresponding ordering into spiked lower triangular form as the *TSS ordering*. An alternative strategy in step 1) of (5.1) is to choose either a minimum-row-count row or a minimum-column-count column, according to whichever gives the minimum bandwidth in the current profile, but we have not explored this.

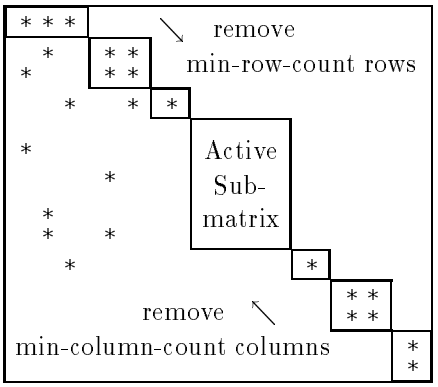

Figure 5.1 Intermediate stage for the Two-sided Algorithm

6. Practical Experience and Conclusions

In this section we report some limited experience on some problems that occurred in practice, and draw some tentative conclusions. The numerical experience arises in applications of a linear programming method to the SOL test problem set (see [8]). There are six matrices that we consider, derived from running our LP code on the test problems. The actual SOL test problems from which the matrices are derived are respectively etamacro, bandm, stair(two matrices) and pilot(two matrices). The first two columns in each table of results give the dimension n of each matrix and the number of nonzeros in the original matrix.

Tables 6.1 and 6.2 give a comparison of various different tie-break rules for step 1) of algorithm (2.3), together with results for the Two-sided ordering described in Section 5. These tables provide purely structural measures for the different forms that can be

derived from the lower Hessenberg form, as described in Section 2. The column headed *Spikes* gives the total number of spikes in the corresponding spiked lower triangular form. The column headed *Border* gives the width of the border, and hence the dimension of the Schur complement, in the corresponding bordered block lower triangular form. The column headed *B-width* gives the bandwidth of the lower Hessenberg form, that is the number of diagonal bands immediately above the main diagonal which contain nonzero elements (a tridiagonal matrix would have a unit bandwidth). This measure is equivalent to the quantity *Local* introduced in [11].

These results give some insight into various issues raised in previous sections. First we consider whether there is any indication that the extra calculation involved in look-ahead gives rise to a superior ordering in any sense. We see that the Hellerman-Rarick tie-break used by both P^4 and P^5 gives better results for all these measures on the 335 problem, but otherwise is not superior. It is interesting that the Stadtherr-Wood tie-break used by the SPK1 algorithm does best on the larger problems, and particularly so on the 1111 problem. There is not much to choose between any of the algorithms on the smaller problems. Thus we tentatively conclude that there is not much to be gained by look-ahead, and that the SPK1 algorithm is to be preferred on account of its simplicity. This conclusion is backed up by the results in [12] which follow a similar pattern. However it is interesting to see that an arbitrary resolution of ties, by taking the first eligible candidate, also does well, and is of course even more simple and efficient to operate. This suggests for example that other criteria might be successfully used to break ties, for example by selecting the block on the grounds of numerical stability.

Problem		HR tie-break			SW tie-break		
n	Nonzeros	Spikes	Border	B-width	Spikes	Border	B-width
113	309	5	4	4	5	5	4
116	592	32	20	12	28	19	12
225	2034	71	63	38	73	61	31
335	3419	81	74	33	98	89	44
366	1781	130	80	48	123	75	48
473	2368	158	100	54	149	93	54
723	8027	148	107	72	122	104	57
1111	16090	207	180	107	161	149	62

Table 6.1 Comparison of tie-break rules

Problem		First candidate tie-break			Two-sided ordering		
n	Nonzeros	Spikes	Border	B-width	Spikes	Border	B-width
113	309	5	5	4	5	5	3
116	592	22	14	7	25	12	10
225	2034	61	54	32	58	54	54
335	3419	94	87	44	90	83	74
366	1781	127	81	48	100	59	31
473	2368	149	93	54	113	73	31
723	8027	136	115	55	134	112	52
1111	16090	183	167	46	203	179	114

Table 6.2 Comparison of tie-break rules

Next we consider the evidence relating to the Two-sided algorithm described in Section 5. It had been hoped that this algorithm would produce orderings in which the bandwidth and the border would be smaller, particularly on larger problems. Unfortunately the evidence here does not support such a conclusion. We shall also see in Tables 6.3, 6.4 and 6.5 that the fill-in due to the TSS ordering is disappointingly high, which reinforces these conclusions based on the structural information. Unless these results are atypical, we see no merit in pursuing this idea any further.

Next we consider how much fill-in is generated when these algorithms are used, and our remarks are relevant mainly to the spiked lower triangular form. An ideal situation for this structure would be to use Gaussian elimination without interchanges, because this would be expected to give relatively little fill-in. However, as observed above, the elimination algorithm might break down due to the occurrence of a zero pivot. Thus in the first instance we give numerical results that are obtained using the following minimal column pivoting strategy, that is *when a zero pivot occurs, use the nonzero entry closest to the diagonal as the pivot.* Table 6.3 gives the number of interchanges performed to avoid a zero pivot, and the amount of fill-in that occurred. The arithmetic was performed in single precision on a SUN 4 workstation. In the two cases indicated the elimination broke down due to no nonzero pivot being available with the above strategy. This was due to catastrophic growth which occurred in most of the factorizations. Indeed, for the matrix of order 1111 overflow occurred in the calculation of some entries.

Problem		Interchanges			Fill-in		
n	Nonzeros	HR	SW	TSS	HR	SW	TSS
113	309	1	1	1	176	174	179
116	592	7	5	5	517	459	373
225	2034	6	24	13	2929	2002	5836
335	3419	5	*32	21	3891	3731	8574
366	1781	46	47	31	2659	3419	3080
473	2368	64	57	33	3993	3656	4164
723	8027	71	47	66	15046	14955	18146
1111	16090	114	59	**97	30165	20805	62319

$*$ Breakdown in stage 335 of the elimination
$**$ Breakdown in stage 1022 of the elimination

Table 6.3 Fill-in with minimal pivoting

These results give some indication of the least fill-in that can be expected by using these ordering algorithms. When compared with the results for the Markowitz strategy (see the column headed MA28 in Table 6.5) it is seen that Markowitz already gives less fill-in on almost all the problems. Thus we cannot expect ordering algorithms to be competitive as regards this measure. Of course it is clear that the minimal pivoting strategy is entirely unsatisfactory as regards numerical stability. Therefore, as indicated in Section 4, we have to consider alternative pivot strategies which provide increased stability at the cost of additional fill-in.

An issue raised in Section 2 is the extent to which fill-in is affected by the method of breaking ties in algorithms (2.3) and (5.1). We consider two alternatives to breaking ties by removing the maximum number of entries (as in the SPK1 algorithm). The first of these is the arbitrary tie-break rule in which the first available candidate is chosen and in the second the minimum number of entries is removed (in the spirit of Markowitz). For these criteria the fill-in in the factorization using the minimal pivoting strategy is given in the columns headed *First* and *Min* respectively in Table 6.4. This is done for both the SW and TSS orderings and may be compared with the results in Table 6.3 for the maximum-removal criterion.

The results show that neither of the 'extreme' tie-break rules is preferable to the essentially arbitrary tie-break rule. The vastly improved results with the arbitrary tie-break rule for the two-sided algorithm in Table 6.4 can be explained as follows. In the case when the counts of the minimum-row-count row and minimum-column-count column in step 1) of (5.1) are equal the first of the tied rows was selected. Thus few, if any, columns were selected so the resulting lower Hessenberg form was similar, if not identical, to that obtained using the SPK1 arbitrary tie-break rule. The TSS ordering is seen to be wholly inferior to the SW ordering and is not pursued further.

Since it is not clear whether the SW ordering is better using the maximum-removal or arbitrary tie-break rules, both are used for the experiments below with the ordering corresponding to the latter identified as SW0.

Problem		SW		TSS	
n	Entries	First	Min	First	Min
113	309	271	337	271	319
116	592	367	670	354	769
225	2034	2722	3038	2986	6038
335	3419	3212	5056	7803	14297
366	1781	3730	4401	3537	3817
473	2368	4151	5966	4143	7091
723	8027	14024	20247	14625	29247
1111	16090	23466	45955	24555	65029

Table 6.4 Alternative tie-break criteria

Since the numerical properties of the minimal pivoting strategy are unsatisfactory, we used the LU1SOL threshold pivoting strategy for the elimination in order to obtain numerical stability whilst attempting to preserve the a priori ordering as much as possible. The results in Table 6.5 give a comparison of the fill-in caused when Gaussian elimination with this pivoting strategy is applied to the spiked triangular form obtained by the HR, SW and SW0 ordering algorithms.

n	Nonzeros	HR	SW	SW0
113	309	198	196	262
116	592	869	750	440
225	2034	3270	2671	2838
335	3419	5028	5223	4393
366	1781	2993	4694	4467
473	2368	7878	5735	6342
723	8027	28941	25172	25550
1111	16090	57053	39100	48516

Table 6.5 Fill-in with LU1SOL pivoting ($u = 0.1$)

There is a clear increase in fill-in when the figures in Table 6.5 are compared with those in Tables 6.3 and 6.4 for the corresponding ordering with the minimal pivoting strategy. This is the price which must be paid for numerical stability. No unacceptable growth was detected on any of the test problems. Smaller values of u cause significant increases in growth and our results support the usual choice of $u = 0.1$.

A possible explanation for this significant increase in fill-in might be that the large number of column interchanges demanded by the LU1SOL strategy destroys the spike column assignment of the ordering algorithm. If the column ordering cannot be preserved then a local column pivoting strategy which aims to reduce fill-in is preferable. To test this conjecture, the factorization of the matrices was repeated using the MINCOL strategy. The results are given in Table 6.6 which also shows the outcome of the MINCOL elimination procedure being applied to the unordered matrix and to the matrix with its rows ordered simply by increasing row count. The latter approach to

ordering and factorization is given by Duff and Reid [4]. The final column in the table gives the fill-in caused when the Markowitz procedure MA28 (as used by the double precision NAG routine F01BRF) is applied to the matrix. In all cases the threshold parameter $u = 0.1$ is used.

n	Nonzeros	None	ROW	HR	SW	SW0	MA28
113	309	249	181	182	186	244	139
116	592	598	567	700	602	403	248
225	2034	5761	2624	2877	2577	2404	1260
335	3419	10364	4222	4541	4223	3904	2181
366	1781	2472	2023	2401	3445	2737	1267
473	2368	4167	3168	4609	4959	4769	1726
723	8027	24005	26341	21700	20364	19260	10064
1111	16090	46668	100898	48379	32858	35835	21874

Table 6.6 Fill-in with MINCOL pivoting and Markowitz ($u = 0.1$)

By comparing the results for LU1SOL and MINCOL in Tables 6.5 and 6.6, it is clear that the MINCOL strategy reduces the fill-in for the three spiked lower triangular forms. Unfortunately, with the exception of the 1111 matrix, this fill-in is little, if at all, better than that incurred when the MINCOL strategy is applied to the matrix ordered by increasing row count. The fill-in incurred when Markowitz pivoting is used is significantly lower than that for the MINCOL strategy applied to any a priori ordering.

The evidence of Tables 6.5 and 6.6 supports the view that when a threshold parameter is used that is sufficiently large to give acceptable stability, the number of interchanges is such that the column assignment of the ordering algorithms is somewhat irrelevant. Hence they are effectively reduced to the status of row ordering algorithms. A further conclusion from these results is that the row permutation determined by the three tie-break rules may be no better than that determined simply by increasing row count.

The conclusions which we offer are based on the results obtained from experiments on a small number of test matrices in an area of numerical linear algebra which is notoriously problem-dependent. Some would point to the density of the the larger problems and say that a matrix with around 10 entries per column is at best 'not *very* sparse'. Certainly the size of the diagonal blocks and border in the P^5 bordered block triangular form appears greater than that observed by Arioli and Duff in [1] so perhaps our problems are not amenable to exploitation by more sophisticated ordering algorithms. We plan to extend these results to a wider selection of test problems at a future date.

7. References

[1] Arioli M., Duff I.S., Gould N.I.M. and Reid J.K. (1987). The practical use of the Hellerman-Rarick P^4 algorithm and the P^5 algorithm of Erisman et al. A.E.R.E. Harwell Report CSS 213.

[2] Arioli M. and Duff I.S. (1988). Experiments in tearing large sparse systems. A.E.R.E. Report CSS 217.

[3] Duff I.S. (1978). On algorithms for obtaining a maximum transversal. A.E.R.E. Report CSS 49.

[4] Duff I.S. and Reid J.K. (1974). A comparison of sparsity orderings for obtaining pivotal sequences in Gaussian elimination. *J. Inst. Maths. Applics.*, **14**, 281-291.

[5] Duff I.S. and Reid J.K. (1978). Algorithm 529: Permutations to block triangular form. *A.C.M. Trans. Math. Software*, **4**, 189-192.

[6] Duff I.S., Erisman A.M. and Reid J.K. (1986) *Direct Methods for Sparse Matrices.* Oxford Science Publications, Oxford.

[7] Erisman A.M., Grimes R.G., Lewis J.G. and Poole W.G. (1985). A structurally stable modification of Hellerman-Rarick's P^4 algorithm for reordering unsymmetric sparse matrices. *S.I.A.M. J. Numer. Anal.*, **22**, 369-385.

[8] Fletcher R. and Hall J.A.J. (1990). Towards reliable linear programming, in *Numerical Analysis 1989*, D.F.Griffiths and G.A.Watson (Eds.) Pitman Research Notes in Mathematics 228, Longman, Harlow.

[9] Hellerman E. and Rarick D.C. (1972). The partitioned preassigned pivot procedure (P^4), in *Sparse Matrices and their Applications* D.J.Rose and R.A.Willoughby (Eds.) Plenum, New York.

[10] Markowitz H.M. (1957). The elimination form of the inverse and its application to linear programming, *Management Sci.*, **3**, 255-269.

[11] Stadtherr M.A. and Wood S.E. (1984). Sparse matrix methods for equation based chemical process flowsheeting - *I* Reordering phase, *Computers and Chemical Engineering*, **8**, 9-18.

[12] Stadtherr M.A. and Wood S.E. (1984). Sparse matrix methods for equation based chemical process flowsheeting - *II* Numerical phase, *Computers and Chemical Engineering*, **8**, 19-33.

[13] Tarjan R.E. (1972). Depth first search and linear graph algorithms, *S.I.A.M. J. Comput.*, **1**, 146-160.