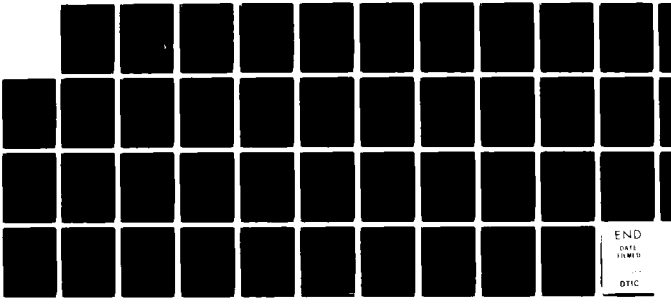
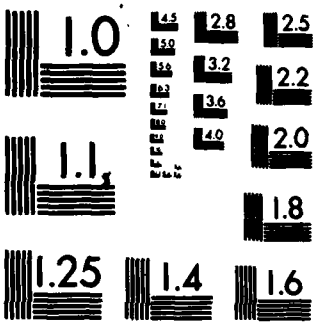
AD-A129 344 A STRUCTURALLY STABLE MODIFICATION OF HELLERMAN-RARICK'S P4 ALGORITHM FOR..(U) BOEING COMPUTER SERVICES CO SEATTLE WA MATHEMATICS AND MODELI

1/1

UNCLASSIFIED .. A M ERISMAN ET AL. APR 83 MM-3 F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFOSR-TR- 83-0479

MATHEMATICS & MODELING TECHNICAL REPORT

BOEING COMPUTER SERVICES COMPANY

MM-3

A Structurally Stable Modification of Hellerman-Rarick's P4 Algorithm for Reordering Unsymmetric Sparse Matrices

A. M. Erisman

R. G. Grimes

J. G. Lewis

W. G. Poole, Jr.

April 1983

Energy Technology Applications Division
G-7500, M/S 9C-01
P.O. Box 24346
SEATTLE, WASHINGTON 98124

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 83-0479	2. GOVT ACCESSION NO. ADA129344	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A STRUCTURALLY STABLE MODIFICATION OF HELLERMAN-RARICK'S P^4 ALGORITHM FOR REORDERING UNSYMMETRIC SPARSE MATRICES		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL 6. PERFORMING ORG. REPORT NUMBER MM-3
7. AUTHOR(s) A.M. Erisman, R.G. Grimes, J.G. Lewis and W.G. Poole, Jr.		8. CONTRACT OR GRANT NUMBER(s) F49620-81-C-0072
9. PERFORMING ORGANIZATION NAME AND ADDRESS Mathematics & Modeling Unit Boeing Computer Services Company Seattle WA 98124		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A3
11. CONTROLLING OFFICE NAME AND ADDRESS Mathematical & Information Sciences Directorate Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE APR 83 13. NUMBER OF PAGES 47
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Sparse matrices, reordering algorithms, Hellerman-Rarick, P4, unsymmetric matrices, Gauss elimination.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Partitioned Preassigned Pivot Procedure (P^4) of Hellerman and Rarick reorders unsymmetric sparse matrices in order to decrease computation and storage requirements when solving sparse systems of linear equations. It is known that the algorithm, when applied to matrices which are not structurally singular, can generate intermediate matrices which are structurally singular, causing a breakdown in the elimination process. In this paper, the authors present the algorithm in a structured, top-down, form and explain several of the problems which may occur. The authors then define a modification (CONT.)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONTINUED: of the algorithm to treat the difficulties. This revised version of the algorithm will never produce structurally singular intermediate matrices if the original matrix is not structurally singular. Test results with this modified algorithm show that it is as effective as the Markowitz algorithm as a preordering when the block structure of the new algorithm is recognized and used.

A STRUCTURALLY STABLE MODIFICATION OF HELLERMAN-RARICK'S P^4 ALGORITHM FOR REORDERING UNSYMMETRIC SPARSE MATRICES*

A. M. ERISMAN+, R. G. GRIMES+, J. G. LEWIS+ AND W. G. POOLE, JR.++

Abstract. The Partitioned Preassigned Pivot Procedure (P^4) of Hellerman and Rarick reorders unsymmetric sparse matrices in order to decrease computation and storage requirements when solving sparse systems of linear equations. It is known that the algorithm, when applied to matrices which are not structurally singular, can generate intermediate matrices which are structurally singular, causing a breakdown in the elimination process. In this paper the authors present the algorithm in a structured, top-down, form and explain several of the problems which may occur. We then define a modification of the algorithm to treat the difficulties. This revised version of the algorithm will never produce structurally singular intermediate matrices if the original matrix is not structurally singular. Test results with this modified algorithm show that it is as effective as the Markowitz algorithm as a preordering when the block structure of the new algorithm is recognized and used.

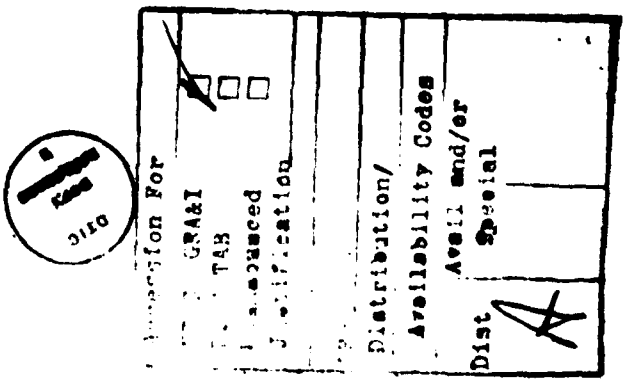
---

*This paper is a result of work supported by the U. S. Air Force Office of Scientific Research under contract F49620-81-C-0072.

+Mathematics and Modeling Unit, Boeing Computer Services Company, Seattle, Washington 98124.
++Pacific Analysis & Computing Corporation, Bellevue, Washington 98004.

1. Introduction. In 1971, Hellerman and Rarick [8] introduced the Preassigned Pivot Procedure (P^3) for ordering rows and columns of *unsymmetric* sparse matrices to preserve sparsity in LU factorizations. The algorithm was designed for use with the matrices encountered in linear programming codes. In 1972, they published a modification called the Partitioned Preassigned Pivot Procedure (P^4) [9]. P^4 added an initial stage of permuting the matrix to block lower triangular form to be followed by applying P^3 to each of the irreducible diagonal blocks.

The P^4 algorithm has attained considerable popularity in application codes, particularly for linear programming problems, even though, in 1974, Westerberg [18] displayed a nonsingular matrix which encountered a zero pivot during Gaussian elimination using the P^4 ordering. Lin and Mah also encountered difficulties with P^4 and suggested an alternative ordering [10,11,12]. (Despite these difficulties, P^4 codes remain in use for linear programming problems [1,14,15,16].) Westerberg, Lin and Mah are chemical engineers interested in optimization problems encountered in chemical process modeling. The matrices from linear programming and from chemical process modeling have in common the properties that they are very sparse and their structures are far from symmetric, yet the published behavior of the P^4 algorithm on matrices from these two fields is dramatically different.



Even though the P^4 algorithm has been used extensively, it has never been analyzed or compared with other reordering algorithms in a detailed, careful study. The structural singularity problem has not been clarified and the possible effect on sparsity of pivoting for numerical stability is not well understood. A classification of what types of matrices can be reordered "well" by P^4 is unknown. Furthermore, a comparison of the effectiveness of P^4 with the popular Markowitz reordering [3,13] has not been carried out.

In order to answer some of these questions and better understand P^4, the authors have carried out a comprehensive study of P^4. In this paper we report our successes in answering the first of these questions, which leads to a new modification of the algorithm which eliminates the structural zero pivot flaw in the original algorithm. We do not have complete answers to the performance questions; indeed, the modification to the algorithm raises new questions about the implementation of the numerical factorization based on this reordering. Therefore, the performance results in this paper are only preliminary and more extensive results will be reported later.

This paper is primarily a report of our understanding of the P^4 algorithm and of its behavior, particularly in situations where it produces a necessarily zero pivot in the factorization. In section 2, a new and precise, yet simple, statement of the P^4 algorithm is given. Section 3 provides definitions of critical terms needed to explain the weakness of the algorithm, which are illustrated by the sample

matrices in section 4 which cause P^4 to fail. An analysis of these counterexamples uncovers a block structure which is useful for analyzing P^4. This structure leads to a new modification of P^4 which does not fail unless the matrix is structurally singular and which still retains sparsity of the matrix. The modified algorithm is presented in section 5; section 6 contains a discussion of implementation considerations for the new modification and some test results.

2. **The Partitioned Preassigned Pivot Procedure (P^4).** The purpose of a reordering algorithm is to rearrange rows and columns of the original sparse matrix so that, when Gaussian elimination is applied to the reordered matrix, the storage and the number of arithmetic operations is less than if Gaussian elimination were applied to the original matrix. Attaining either minimal storage or minimal work is an extremely difficult combinatorial problem, for which it is believed that practical algorithms cannot exist. The best that is possible in practice is to apply heuristic reordering algorithms which attempt to reduce these measures of cost. For example, probably the most simple and most common reordering algorithms are designed to rearrange rows and columns so that all of the nonzeros are clustered along the main diagonal. Such algorithms are called band- or profile-reduction algorithms. On the other hand, Hellerman and Rarick's P^4 algorithm is designed to produce, by row and column permutations, a bordered block triangular form (BBTF) of the original matrix:

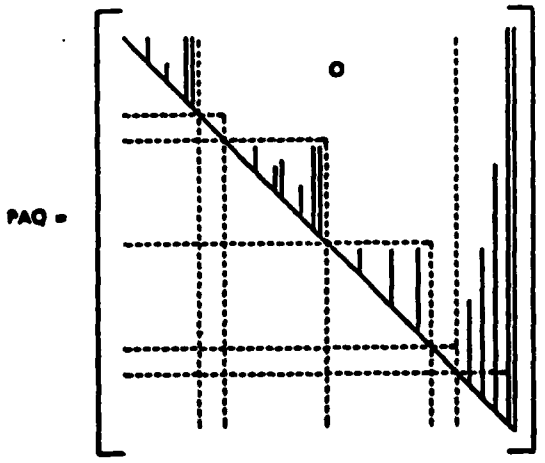


Figure 2.1

Gaussian elimination often introduces new nonzeros or fill in the decomposed matrix. A bordered block triangular form may be a desirable form for performing Gaussian elimination on a sparse matrix because the fill which results is localized. There may be fill throughout the border. Fill in the block lower triangular portion of the matrix occurs locally; the fill within and beneath a diagonal block depends only on the nonzero pattern of those columns, but not on any other columns in the matrix. Indeed, if the matrix can be written in bordered triangular form, that is, with all leading diagonal blocks of order one, there will be no fill except in the border. Further, implicit block factorization schemes [4,7] provide a way to factor such matrices while requiring additional space only for the fill in the diagonal blocks. The bordered block triangular form is also useful in the context of large-order linear programs, where the column orientation of the border meshes well with input/output and updating requirements [16].

The reasons which argue for the desirability of a bordered block triangular form apply equally well when the border is empty, that is, when the matrix is in block triangular form. Non-trivial block triangular forms exist only for reducible matrices. Fortunately it is well-understood how to find such forms; the algorithm by Tarjan [17] as implemented by Duff and Reid [5,6] is a very efficient tool for this purpose. The major difference between the original P^3 algorithm and the P^4 algorithm is that the latter algorithm is essentially the P^3 algorithm applied only to the irreducible diagonal blocks in the

finest block triangular form. Analyzing only the diagonal blocks rather than the entire matrix reduces the size of the matrices analyzed and thereby reduces the computational complexity. Knowing that the diagonal matrices are irreducible simplifies the reordering algorithm. The resulting reordered form of the matrix is a block triangular form in which each diagonal block is itself in bordered block triangular form. We assume herein that the P^3 algorithm is applied only to irreducible matrices; in practice these will be the diagonal blocks of a block triangular form.

The overall objective of P^4 is to find a row and column ordering such that the factors of the resulting bordered block triangular form are sparse. The immediate goal of the heuristic reordering is to choose a small number of columns to be in the border so that the remaining subsystem will be in block triangular form with a large number of small diagonal blocks. The border columns are called spikes. P^4 chooses the columns to be spikes, to be put in the border, on the basis of a tally function which keeps track of the number of nonzeros in each row. The P^4 heuristic uses the tally function to extend an ordering which has assigned the leading rows and columns of a bordered block triangular form, as will be described below.

Suppose that $i-1$ rows and columns of a sparse matrix have been assigned as the leading block(s) in a bordered block triangular form and that additional columns in the matrix have been assigned as spikes. We may extend this form an additional row and column at the

possible cost of adding one or more spikes to the border as follows. Consider only the active entries in the matrix, that is, the nonzero entries lying in the intersection of rows and columns which have not yet been assigned. If any row has only one nonzero entry, that entry can be brought to the (i,i) position by permuting rows and columns; the remaining part of the row will be zero and so the block triangular form can be extended by one row. If there is no such row (and there will not be any such row at the first step), we must choose additional spikes. The tally function, which keeps track of the number of nonzeros in each row, indicates those rows of minimum row-count. P^4 chooses as a new spike a column which reduces the minimum row count (ties are broken in a manner which is described in the algorithm). This process continues until the minimum row count is one. When the minimum row count is reduced to one, the rows and columns are permuted so that the nonzero in a row with only one nonzero is placed along the main diagonal. This is the general idea of the P^4 algorithm. Note that at any step, we will choose exactly one fewer spike than the initial minimum row-count.

As an example, consider the stage of the algorithm represented by the matrix in Figure 2.2 (where * represents a nonzero)

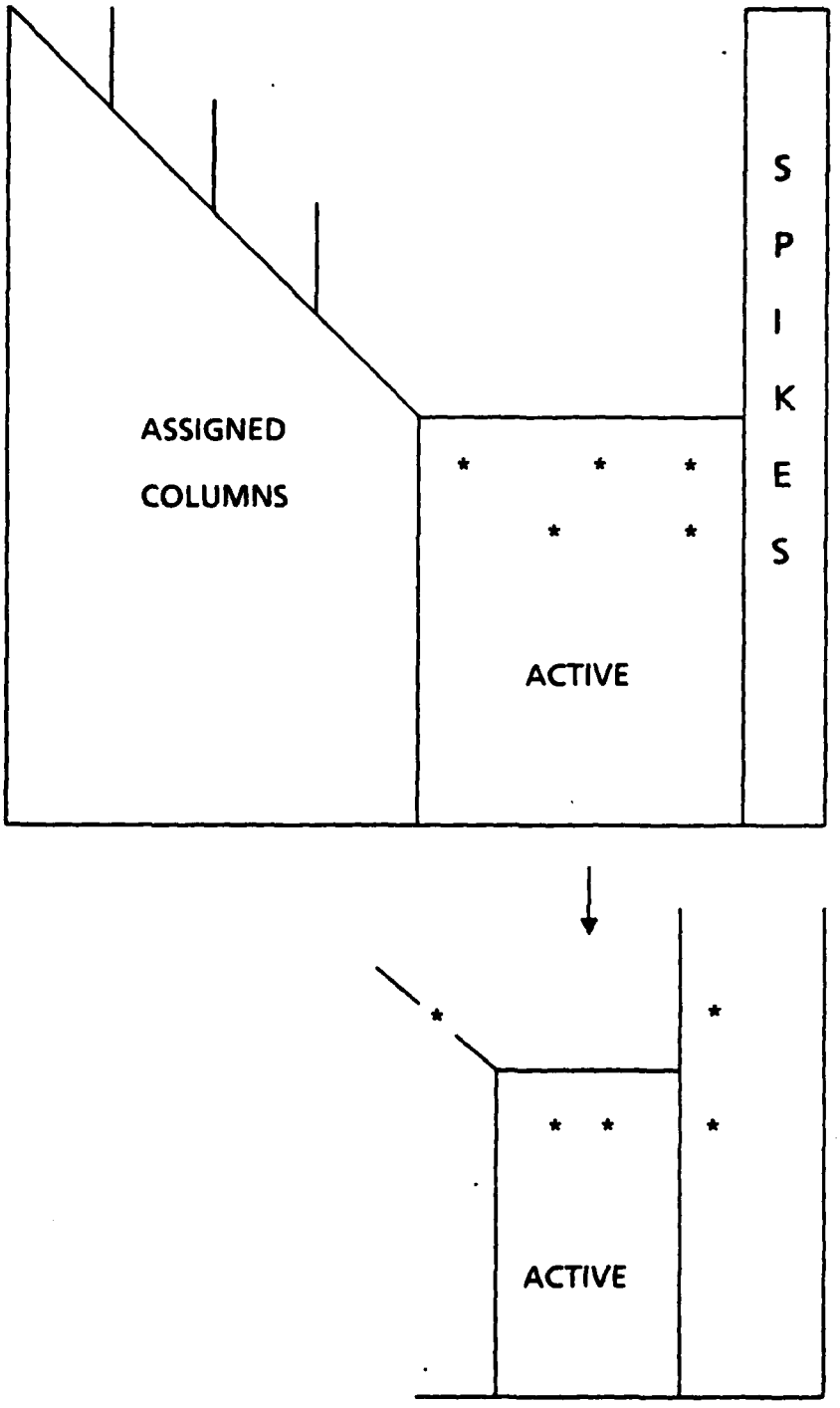


Figure 2.2

At this point, there is a row with two nonzeros in the "active" part of the matrix and another row with three nonzeros. If the last column in the active part is chosen as the next spike to be added to the border, then one row is now a singleton (contains only 1 nonzero). By interchanging appropriate rows and columns, this nonzero can be placed in the (1,1) position of the active part of the matrix and we have added one more row and column to the block triangular part.

The original presentation of the algorithm is quite complex even though the major thrust of the heuristic is simple. This is due to three factors. First, the heuristic is made more sophisticated by attempting to anticipate the next step in the extension of the triangular form: a good choice for a spike at one step may reduce the number of spikes required at the next step. This results in an purposeful, but complicated, tie-breaking strategy for choosing spikes. Second, it is often the case that the overall fill can be reduced if spikes can be moved left from the border into diagonal blocks in the triangular portion of the form. This is possible whenever more than one row becomes a singleton row when a new spike is assigned. The Hellerman-Rarick algorithms incorporate this idea, which has the effect of enlarging the diagonal blocks while reducing the size of the border. The size of the enlarged diagonal blocks somewhat obscures the BBTF structure and reduces the effectiveness of implicit block solvers, but moving the spikes left usually does reduce the fill in an explicit factorization. Last, the original presentation of the algorithm predates the widespread use of

structured languages in scientific programming; a presentation in a structured language simplifies understanding the algorithm dramatically. For this reason, we now give a precise description of the Hellerman-Rarick P^4 algorithm in a PASCAL-like language.

The algorithm appears as four procedures. Procedure HELLERMAN_RARICK_P4 is the main driver which begins by putting the matrix into block triangular form, each of whose diagonal blocks are irreducible. Next the procedure applies a simplified version of P^3 to each (irreducible) diagonal block. The second level procedure, APPLY_P3_TO_DIAGONAL_BLOCK, indicates how a stack of spikes is created as the minimum row count is decreased at each stage. The third level procedure, CHOOSE_A_GOOD_COLUMN_TO_REMOVE, describes the rules for choosing the spike to be removed from the active matrix and placed in the border. The other third level procedure, ASSIGN_IT_AND_POSSIBLY_SOME_SPIKES, determines which columns, including some spikes, are to be assigned to the block triangular form.

```
PROCEDURE HELLERMAN_RARICK_P4;
   BEGIN
   OBTAIN_IRREDUCIBLE_BLOCK_FORM;
   [  Use Tarjan's algorithm or equivalent.  ]
   FOR I := 1 TO NUMBER_OF_DIAGONAL_BLOCKS DO
         APPLY_P3_TO_DIAGONAL_BLOCK(I);
   END;
```

```
PROCEDURE APPLY_P3_TO_DIAGONAL_BLOCK(I);
[ During the application of P3 to the diagonal block, we speak of an "active"
submatrix.  Initially all rows and columns in the diagonal block are active.
Columns become inactive by being chosen to be spikes and/or by being assigned
into final position in the nested block bordered triangular form.  Rows
become inactive by being assigned to final position.  The row-counts in the
procedures which follow refer only to the active submatrix. ]
  BEGIN
  REPEAT
    [ Remove columns from the active matrix (and call them spikes) until
    the triangular part of the bordered form can be extended, i.e., until
    the row-count of some row in the active matrix is reduced to one. ]
    WHILE  MIN_ROW_COUNT > 1  DO
      BEGIN
      CHOOSE_A_GOOD_COLUMN_TO_REMOVE;
      PUT_IT_ON_THE_SPIKE_STACK;
      END;
    [ Extend the triangular portion of the bordered triangular form by
    assigning a column which has the only nonzero in some row of row-count
    one, and perhaps also assigning some spikes from the stack.  Assign as
    many rows as columns. ]
    CHOOSE_A_GOOD_COLUMN_TO_REMOVE;
    ASSIGN_IT_AND_POSSIBLY_SOME_SPIKES;
  UNTIL
    All rows and columns are assigned
  END;
```

```
PROCEDURE  CHOOSE_A_GOOD_COLUMN_TO_REMOVE;
   [  Choice is based on removing a column which locally promotes continuing
      the triangular form as much as possible, that is, reduces the row-
      counts of as many minimum row-count rows as possible.  Tiebreaking part
      of the heuristic is invoked only when there are several columns which
      reduce a maximum number of minimum and low row-count rows. ]
   BEGIN  [ MIN_ROW_COUNT := minimum row count in active submatrix. ]

   CANDIDATES := set of active columns which maximally intersect rows with
                 minimum row-count;

   IF  more than one candidate  AND  all candidates intersect only a single
       minimum-count row  THEN
      BEGIN
      NEXT_LARGER_ROW_COUNT := the second smallest row-count of rows which
                               intersect CANDIDATES;
      CANDIDATES := columns in CANDIDATES which maximally intersect rows
                    of row-count NEXT_LARGER_ROW_COUNT;
      END;

   Choose column from CANDIDATES which has max. number of nonzero entries;
   [  We assume that columns are chosen in order of decreasing index.  ]

   END;
```

```
PROCEDURE  ASSIGN_IT_AND_POSSIBLY_SOME_SPIKES;
   BEGIN  [ Assign a selected column and a nonzero entry in a singleton row
      to the next diagonal position.  For each additional singleton nonzero
      in the selected column we can also remove a spike column from the stack
      and assign it. ]
   Q :=  number of nonzero entries of the selected column which are the only
         nonzeros in their corresponding rows of the active matrix.
   J := column index of chosen column;
   I := row index of some singleton nonzero entry in column J;
   ASSIGN (I, J);
   [  IF  Q > 1  THEN   ]
   FOR  INDEX := 2 TO Q  DO
      BEGIN
      K := column index of spike on top of stack;
      I := row index of some unassigned singleton nonzero entry in column J;
      ASSIGN (I, K);
      END;
   END;
   [ Assume that singleton rows are assigned in order of increasing index.]
```

3. Singularity, Structural Singularity and Structural Stability. The P^4 algorithm reorders sparse matrices in an attempt to reduce the computational requirements for Gaussian elimination. It does so knowing only where nonzero entries are found in the matrix, but not knowing their values and without knowing how the matrix will be affected by fill. Equivalently, the P^4 algorithm is an algorithm for numbering or labelling the nodes of a directed graph, where the nonzero entries correspond to edges of the graph. This numbering prescribes the row and column interchanges which take place before the numerical decomposition or elimination process occurs. Nowhere in the published definition of the algorithm is there any suggestion that additional interchanges are permitted: It is an algorithm for permuting a sparse matrix into a form upon which Gaussian elimination proceeds without any interchanges, whence the name "preassigned pivot procedure". It is a prescription which is the same for all sparse matrices which have a common graph, irrespective of the numerical values assigned to the entries in the matrix or the edges in the graph.

It is a fundamental result that Gaussian elimination may be used to perform an LU decomposition of an arbitrary nonsingular matrix if and only if rows and/or columns are interchanged when necessary to avoid division by zero. It is, of course, also generally necessary in numerical practice to provide for additional interchanges to prevent numerical instabilities. However, the use of the P^4 algorithm as a preassigned reordering of sparse matrices requires further discussion

of the strictly mathematical or symbolic requirements for interchanges.

The graph of a sparse matrix is said to be structurally singular if there is no assignment of nonzero values to the edges in the graph such that the resulting matrix is nonsingular. Equivalently, the original matrix and all other matrices which have nonzeros in the same locations are singular matrices. For example, any matrices with identically zero rows or columns are singular, as are all three by three matrices whose nonzeros fall only in the positions indicated in Figure 3.1. The graph of any such matrix is structurally singular.

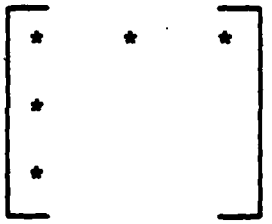


Figure 3.1

A useful criterion for detecting structurally singular graphs is the complete transversal criterion: a graph is structurally singular if and only if there is no column permutation which gives a complete assignment or transversal, that is, reorders the matrix so that the diagonal entries in the permuted matrix are all nonzero [2]. Given a complete transversal it is clear that prescribing the numerical value one to the nonzeros which lie on the diagonal of the permuted matrix

and the numerical value zero to all other nonzeros results in a matrix which is a permutation of the identity and is evidently nonsingular.

The permutation which exhibits a complete transversal provides very useful information about the matrix, but it does not necessarily provide an ordering which is compatible with the preservation of sparsity. Conversely, it is necessary for a reordering which does not allow interchanges in the elimination process to label the graph in such a way that the permuted and modified diagonal elements are nonzero during the elimination. Neither the Markowitz nor the P^4 algorithm are guaranteed to place original nonzero elements on the diagonal, i.e., exhibit a complete transversal. Both algorithms depend on fill due to the modification of elements to produce nonzero entries at the appropriate place on the diagonal at the appropriate time. Both may fail to exhibit a transversal because that goal is incompatible with their sparsity-preserving heuristics.

An algorithm like P^4 has the disconcerting property that we cannot determine, a priori, if Gaussian elimination without interchanges will succeed on the reordered sparse matrix. An ordering for which this is not a concern could be called a structurally stable ordering. A suitable definition of this term is: a reordering algorithm is <u>structurally stable</u> if for every graph which is not structurally singular there is an assignment of nonzeros to the edges of the graph such that (mathematical) Gaussian elimination on the permuted matrix succeeds <u>without interchanges</u>. Conversely, a

reordering scheme is *not* structurally stable if there is some graph which is *not* structurally singular, and yet, for all possible assignments of nonzeros to instances of the graph, Gaussian elimination on the reordered matrix encounters a zero diagonal element.

It should be evident that structural stability does not imply numerical stability, but the lack of structural stability guarantees the existence of problems for which we have complete failure. It has been known for some time that the P^4 algorithm is not a structurally stable reordering; in the next section we present several counterexamples which show this. An analysis of these counterexamples provides a understanding of how the instability arises, and leads, in section 5, to rather minor changes to the algorithm which result in a structurally stable reordering.

4. The P^4 Algorithm is not Structurally Stable. The P^4 algorithm was used in several application programs, particularly for linear programming problems, in the early 1970's. However, in 1974, Westerberg presented a nonsingular matrix which encountered a zero pivot during Gaussian elimination using the P^4 algorithm. The nonzero pattern of the matrix was:

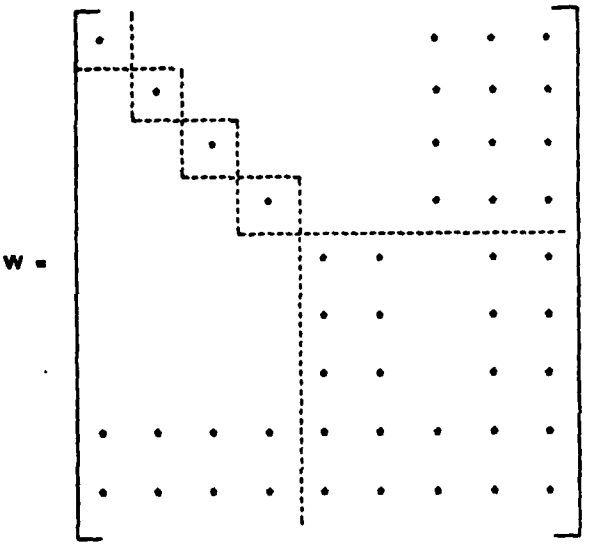


Figure 4.1

Values can be assigned to the nonzero elements which make the matrix nonsingular, but not so that Gaussian elimination can be performed without interchanges. This matrix, as ordered, is not permuted by the P^4 algorithm. Note that the element in the (7,7) position is zero and that it will remain zero during Gaussian elimination.

Clearly the graph is not structurally singular, since interchanging either rows seven and eight or columns seven and eight exhibits a complete transversal. Yet the (7,7) element in the matrix and in its (partial) LU decomposition is always zero. We describe the zero pivot in the (7,7) position as a structurally zero pivot, since it is zero for all possible assignments of nonzeros to the graph. The primary question we pose is whether there are simple modifications to the algorithm which will automatically detect and correct, or prevent, the occurence of such structurally zero pivots. If there are such modifications, how much must the algorithm be modified in order to have such a guarantee? The remainder of this section and the next will address those questions.

The P^4 algorithm naturally partitions a sparse matrix into a block structure which is a refinement of the block structure of the bordered block triangular form. This refined blocking can be understood by examining procedure APPLY_P3_TO_DIAGONAL_BLOCK in section 2. Each execution of the REPEAT loop which is the main body of the procedure creates a new block on the diagonal of the matrix. The procedure call immediately preceding the UNTIL clause defines the block itself. Each such block consists of the assigned column with a singleton row and possibly also some spikes from the stack. The size of the block is the value of the variable Q in procedure ASSIGN_IT_AND_POSSIBLY_SOME_SPIKES, which is the number of rows in the chosen column which have been reduced simultaneously to singletons. For the Westerberg matrix, the blocks are indicated in Figure 4.1.

One by one blocks in the structure are the result of the ordering heuristic assigning to the next diagonal position the nonzero element in a row which has exactly one nonzero entry in the active matrix. It is clear that structurally zero diagonal elements can occur only in blocks of order greater than one. The fact that structurally zero pivots only occur in larger diagonal blocks suggests an obvious modification: allow row interchanges, but only among rows in a single diagonal block. (The ordering of rows within such a block is not specified in the original description of the algorithm.) Such interchanges do not change the block structure which the ordering induces, but they do allow us to decompose matrices in which none of the diagonal blocks are structurally singular.

The use of row interchanges within diagonal blocks removes Westerberg's matrix as a *counterexample*. Unfortunately there are other difficulties with this approach. One difficulty is that the diagonal block may have fill from earlier blocks in the columns which were removed from the spike stack. Knowing that a diagonal block is not structurally singular may require knowledge of these fill elements and their use in the complete transversal. Fill entries are not "free" in the same sense as original nonzero elements, since they are subject to the algebraic constraints which generate them. Do they really demonstrate that a nonsingular instance of the block exists? Consider the two by two block resulting from one step of Gaussian elimination on the three by three example in Figure 3.1: The graph of this block is not structurally singular, but no instance of the three

by three matrix exists such that the resulting two by two block is nonsingular. Despite these difficulties, it can easily be shown that row interchanges within diagonal blocks is sufficient to avoid structurally zero pivots in the final diagonal block. A complete analysis would attempt to show that structurally nonsingular matrices, reordered by the P^4 algorithm, result in diagonal blocks all of which are not structurally singular.

Unfortunately this result is not true. Consider the 8 by 8 matrix whose nonzeros lie in the indicated positions:

$G =$

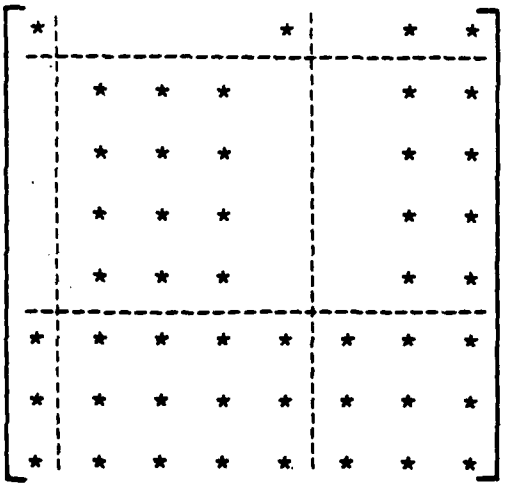


Figure 4-2

As before, the matrix is written in the form which P^4 produces; P^4 does not reorder the matrix. The P^4 algorithm defines three diagonal blocks, the second of which is rank-deficient -- no matter how it is permuted, there will be a zero pivot. This matrix is not structurally singular (interchange rows five and eight to exhibit a complete transversal), but it has a P^4-induced diagonal block which is structurally singular. Thus we must consider modifications which change the induced block structure if we hope to produce a structurally stable modification of the algorithm.

5. The Precautionary Partitioned Preassigned Pivot Procedure (P^5).

The final counterexample of the previous section demonstrated that rank-deficient diagonal blocks may be encountered with the P^4 algorithm, even when the matrices to which it is applied are not structurally singular. Identifying the source of the structurally singular pivot in the counterexample suggests a simple modification of the algorithm which is structurally stable in general.

The difficulty in the counterexample G in Figure 4.2 arises from the fourth column in the second block. Four columns are assigned to this block because four rows are reduced to singletons simultaneously. But the minimum row count at the beginning of the REPEAT iteration which constructs this block was three, which is exactly the number of dense columns in the block. By simply restricting the size of a diagonal block to be no larger than the minimum row count at the beginning of its construction, the problem of structurally zero pivots is solved. Equivalently we restrict the spikes being brought off the stack to being only those which were designated as spikes during the construction of this block. This has the side-effect of leaving more spikes in the final border, but will prevent the occurence of structurally zero pivots.

That this modification is sufficient in general depends on two observations: first, with this modification all diagonal blocks, except possibly the last block, will be dense; second, the final diagonal block, after fill, will never be structurally singular. We shall now justify these claims.

The general structure of a diagonal block assigned by the (unmodified) P^4 algorithm is indicated in figure 5.1. The first column is the column of row singletons, which is dense within the block because the rows in the block are exactly the rows in which the singletons are found. The second group of columns were the topmost columns on the stack of spike columns, the columns added to the stack during the construction of this block. These columns are dense within the block by the nature of the heuristic. At each step some subset of the rows with minimum row count have their row count reduced by one; the final set of singleton rows is exactly the subset of the initial minimum row count rows whose row count is reduced by one at each step. But this implies that each of the final singleton rows has a nonzero entry in each spike assigned during this iteration. These spikes form dense columns within the block. The remaining columns, the third group of columns in Figure 5.1, are the excess spikes removed from the stack, spikes assigned to the stack during earlier iterations. The contributions of these spike columns to the diagonal block are unknown. In the worst case these columns may not add to the rank of the diagonal block, as demonstrated by the counterexample.

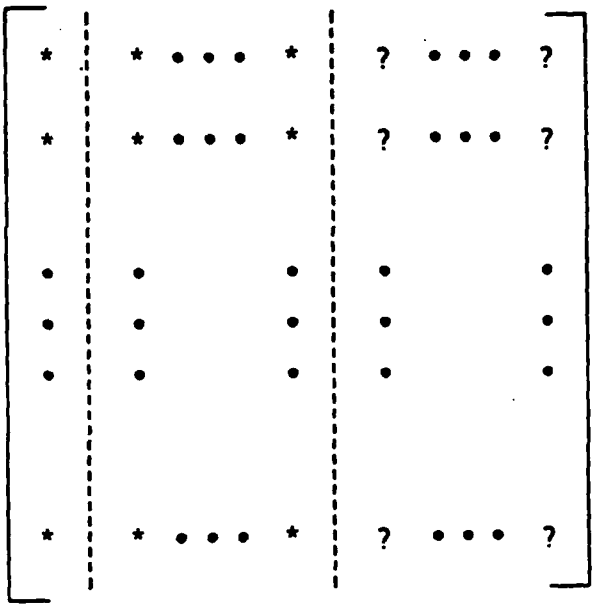


Figure 5.1

The modification to the algorithm prevents assignment of any columns *from this third* group, whether they cause structural singularity in the diagonal block or not. The diagonal block which is produced by the P^5 algorithm is a dense submatrix formed from rows of the leading, dense, columns. A high level description of the modified algorithm is presented below, with the modifications to the P^4 algorithm indicated in italics.

```
PROCEDURE  ASSIGN_IT_AND_POSSIBLY_SOME_SPIKES_MODIFIED;
   BEGIN
   Q := number of nonzero entries of selected column which are the
        only nonzeros in their corresponding rows of the active
        matrix;
   J := column index of chosen column;
   I := row index of some singleton nonzero entry in column J;
   ASSIGN (I, J);
   FOR INDEX := 1 TO MAX (Q-1, number of spikes stacked at this stage)
   DO
      BEGIN
      K := column index of spike on top of stack;
      I := row index of some unassigned singleton nonzero in column J;
      ASSIGN (I, K);
      END;
   END;
```

With this modification, there may be some spike columns remaining on the stack when the algorithm is finished. Thus we also need to modify the top level P^3 procedure:

```
PROCEDURE APPLY_P3_TO_DIAGONAL_BLOCK(I)_MODIFIED;
   BEGIN
   REPEAT
      WHILE  MIN_ROW_COUNT > 1  DO
         BEGIN
         CHOOSE_A_GOOD_COLUMN_TO_REMOVE;
         PUT_IT_ON_THE_SPIKE_STACK;
         END;
      CHOOSE_A_GOOD_COLUMN_TO_REMOVE;
      ASSIGN_IT_AND_POSSIBLY_SOME_SPIKES;
   UNTIL
      All columns are either assigned or designated as spikes;
   ASSIGN_SPIKES_REMAINING_ON_THE_STACK
   END;
```

The other two procedures, HELLERMAN_RARICK_P4 and CHOOSE_A_GOOD_COLUMN_TO_REMOVE, are unchanged for the P^5 algorithm. The new procedure required here, ASSIGN_SPIKES_REMAINING_ON_THE_STACK, will be discussed later.

The matrix that results from the P^5 algorithm is obviously in bordered block triangular form:

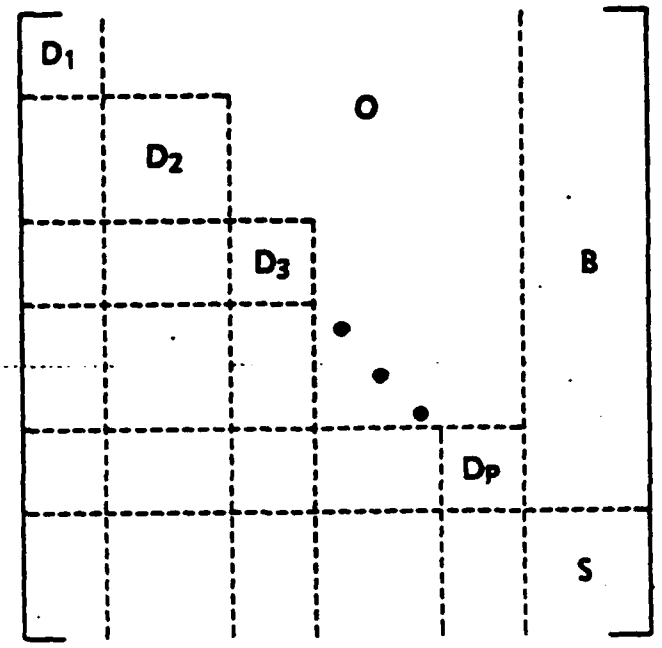


Figure 5.2

All of the leading diagonal blocks, D_j, are dense: they are not structurally singular. The final diagonal block, S, may be structurally singular. However, after the fill from Gaussian elimination, the filled final block, $\hat{S}$, cannot be structurally singular. Showing this fact will demonstrate that our modifications are always effective in removing the problem of structurally zero pivots. It is natural to attempt a proof using characterizations of the graph of $\hat{S}$. However, we encounter the difficulty that the fill elements are dependent on algebraic constraints and so are not free. We side-step these problems by creating a procedure for constructing counterexamples: recall that a graph is not structurally singular if we can assign nonzeros to its edges so that the resulting matrix is nonsingular.

Our procedure requires several tools from linear algebra which address singularity of matrices. The first is the Schur complement theorem, which states that if a nonsingular matrix A is partitioned as

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

where A_{11} is nonsingular, then the Schur complement or Gauss Transform, $\hat{A}_{22}$, of A_{11} in A, is nonsingular. The second result we need is that no perturbation A + E of A is singular if the Euclidean norm of E is less than the smallest singular value of A.

Assume that we have a graph which is already labelled by the P^5 algorithm, and assume that the graph is not structurally singular. We can assign values to the edges of the graph in such a way that the resulting matrix A is nonsingular. Our goal is to show that we can assign values to the nonzeros in such a way that A and all of its leading diagonal blocks D_j are nonsingular. If we can do this much we can apply the Schur complement theorem treating the entire triangular portion as a single block to assert that $\hat{S}$ is not singular. But this demonstrates that the graph of $\hat{S}$ is not structurally singular since we will have shown the existence of a nonsingular instance of the graph. This in turn demonstrates that the P^5 algorithm is structurally stable.

Consider then the assignment of the nonzero values of A. Since A is not structurally singular it has a complete transversal. We begin by assigning the value one to all entries on the transversal and zeros elsewhere: call the resulting matrix A_I. The ordering produced by P^5 does not assign the entries on the transversal to the diagonal of A_I, which means that the diagonal blocks D_j may yet be rank-deficient. The matrix A_I is a permutation of the identity matrix, which implies that all of its singular values are one. Note that a diagonal block D_j is singular if and only if some (or all) of its columns are identically zero. A block D_j of order p and rank $q < p$ contains q columns of a p by p permutation matrix. Since all of the entries of D_j are allowable nonzeros in the graph, we can choose p-q new entries, one in each of the p-q null columns, which would produce a p by p permutation matrix if they were each assigned the value one. Let E_j be the p by p matrix containing the value one-half (or any other nonzero values of magnitude no larger than one-half) in the positions corresponding to these new entries and zeros elsewhere. It is evident that $D_j + E_j$ is nonsingular. Let E be the matrix which is zero except in its leading diagonal blocks, which are E_j. Then the matrix $A = A_I + E$ will have the properties we desire. This matrix is nonsingular, since the Euclidean norm of E is no greater than one-half, smaller than the smallest singular value of A_I. But A also has the property that all of its leading diagonal blocks are nonsingular. It must then be true that $\hat{S}$ is nonsingular.

With this result in hand, we can now state the requirements for the undefined procedure ASSIGN_SPIKES_REMAINING_ON_THE_STACK. The P^5 algorithm

will be structurally stable if this procedure assigns the rows and columns of S in any way which exhibits a complete transversal of $\hat{S}$. One way in which this may be done is to symbolically factor the reordered matrix to obtain the graph of the Schur complement and then apply a transversal-finding algorithm, such as in [6]. Although not necessarily the most efficient algorithm, this does guarantee that the overall algorithm is structurally stable. In practice this final block, $\hat{S}$, may be nearly dense, in which case the factorization may be carried out with the ordering specified a posteriori by a dense factorization algorithm.

6. Discussion of the P^5 Algorithm and Test Results. The ordering produced by P^5 has a block structure which can be put to good use in solving linear equations. In this section we discuss how we might take advantage of this block structure. We present fill statistics for an experimental implementation of a P^5 factorization code for a small number of test matrices, with comparative results for a P^4 code and for the Harwell MA28 implementation of the Markowitz ordering. Finally we indicate how modifications to the algorithm, particularly those to preserve numerical stability, interact with the block structure.

We must use two variations of the usual LU factorization to take full advantage of the special matrix structure which results from the P^5 ordering. The first exploits the reducible form of the matrix, and is applicable to the P^4 and Markowitz orderings as well. The second variation makes use of the special block structure which results from applying the P^5 algorithm to an irreducible matrix. We summarize these two computational techniques and then show how they can be combined effectively for the overall P^5 matrix structure.

Most ordering schemes for unsymmetric matrices begin by finding the finest block triangular or reducible form for the matrix, because the majority of the work in ordering and in factoring the matrix can be confined to the diagonal blocks. Suppose that A is a reducible

matrix and has been permuted to block triangular form as illustrated in figure 6.1. Here each of the blocks A_{ii} is square and irreducible.

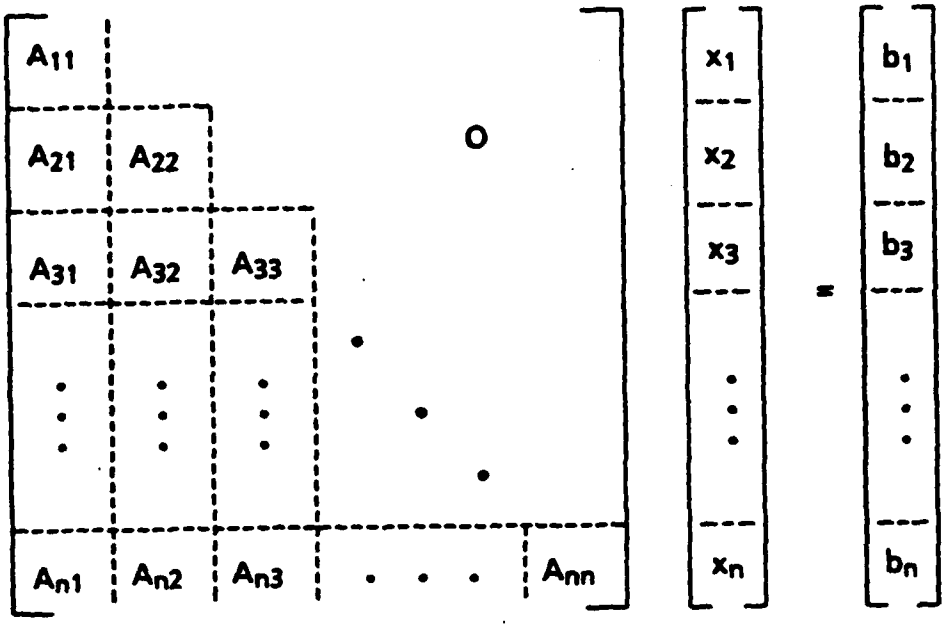


Figure 6.1

The linear system in figure 6.1 can be solved very efficiently by factoring each of the diagonal blocks independently rather than by computing an LU decomposition of A. If we know the original off-diagonal blocks A_{ij}, $j < i$, and also the LU decompositions of the diagonal blocks $A_{ii} = L_{ii}U_{ii}$, then the overall system can be solved by solving in turn

$$L_{11}\, U_{11}\, x_1 = b_1,$$

$$L_{22}\, U_{22}\, x_2 = b_2 - A_{21}\, x_1,$$

and in general

$$L_{ii}\, U_{ii}\, x_i = b_i - \sum_{j=1,i-1} A_{ij}\, x_j$$

We will refer to this technique as the reducible factorization. The reducible factorization clearly saves computation in the factorization phase because no operations are required outside the diagonal blocks. This saves both storage and computation in the solution phase when A is sparse since the off-diagonal blocks in an ordinary block factorization, $L_{ij} = A_{ij} U_{jj}^{-1}$, usually suffer from fill. The filled blocks L_{ij} are more expensive to store and to use as operators than the unfilled original blocks.

The second technique for exploiting block structure will be applied to the irreducible diagonal blocks of the matrix in figure 6.1. Suppose that A_{ii} is nonsingular and has been partitioned as

$$A_{ii} = \begin{bmatrix} \tilde{D} & B \\ C & S \end{bmatrix},$$

where $\tilde{D}$ and S are square, and $\tilde{D}$ is nonsingular. Then an alternate to computing an LU decomposition of A_{ii} is to compute a asymmetric block LU factorization

$$\begin{bmatrix} \tilde{D} & B \\ C & S \end{bmatrix} = \begin{bmatrix} \tilde{D} & 0 \\ C & \hat{S} \end{bmatrix} \begin{bmatrix} I & \tilde{D}^{-1}B \\ 0 & I \end{bmatrix},$$

where $\hat{S} = S - C \tilde{D}^{-1} B$.

This block factorization is the basis for an implicit block factorization. Alan George [7] recognized that this form can be used to solve linear equations without storing $\tilde{D}^{-1}B$, provided that we can solve equations with $\tilde{D}$ and $\hat{S}$. The matrix $\tilde{D}^{-1}B$ is needed to compute $\hat{S}$, but for this its columns can be computed one at a time and then discarded.

Suppose that the LU factors of $\tilde{D}$ and $\hat{S}$ have been computed. Then the system $A_{ii}\, y = f$ can be solved by the following sequence of operations, in which the vectors f, y, and z are partitioned conformally with the partitioning of A_{ii}:

1) Solve $\tilde{D}\, z_1 = f_1$ for z_1;
2) Solve $\hat{S}\, z_2 = f_2 - Cz_1$ for z_2;
3) Set $y_2 = z_2$;
4) Form $w = By_2$;
5) Solve $\tilde{D}\, u = w$ for u;
6) Set $y_1 = z_1 - u$.

Since we did not store $\tilde{D}^{-1}B$, the multiplication by this product is accomplished implicitly by the fourth and fifth steps.

The primary advantage of this approach is that it saves storage, since the fill in the off-diagonal blocks is not saved or stored. It may require additional computation, however, because the steps which realize $\tilde{D}^{-1}B$ implicitly may require more work than if $\tilde{D}^{-1}B$ were stored. We require the solution of two systems involving $\tilde{D}$ instead of one. The implicit form may represent less computation in certain

circumstances, as when B is very sparse and $\tilde{D}^{-1}B$ is not, or when $\tilde{D}$ has special structure which makes solving systems very easy. One such case is when $\tilde{D}$ has a block triangular structure for which we can use the reducible factorization discussed above.

The ordering produced by the P^5 algorithm allows us to put all of these ideas to use simultaneously. The first stage of the algorithm is to find a reducible, block triangular form for the matrix, as in figure 6.1. We can use the reducible factorization at this outer level to confine our factorization to the diagonal blocks A_{ii}. At this middle level of the blocks A_{ii} we can use the implicit block factorization since the P^5 algorithm produces a bordered block triangular form for A_{ii}, as illustrated in figure 5.2. Our notation for the blocks in the implicit factorization partitioning of A_{ii} suggests our intention: we associate the block triangular part of figure 5.2 with $\tilde{D}$ and the other subblocks in Figure 5.2 with the subblocks of the same name in A_{ii}. Since $\tilde{D}$ is in block triangular form we can use the reducible factorization to solve equations with $\tilde{D}$.

Our suggested factorization and solution schemes for the matrix A is to use the reducible factorization for the outermost partitioning, the implicit block factorization for the diagonal blocks A_{ii} and again the reducible factorization for the subblock $\tilde{D}$ within each block A_{ii}. The effect of the first reducible factorization is that fill is confined to the diagonal blocks A_{ii}. The effect of the

implicit block factorization is to confine the fill within A_{ii} to its diagonal blocks $\tilde{D}$ and $\hat{S}$. Using a reducible factorization for $\tilde{D}$ confines the fill within $\tilde{D}$ to its diagonal blocks -- but these are already full. Thus, the only fill in the overall scheme is the fill within the Schur complement blocks for the border, $\hat{S}$. Fill occurs nowhere else in the matrix.

The usual cost of changing the form of the factorization to save space is to increase computation. We have used alternative factorizations at three levels, but the only computation penalty we bear is that we must solve two (not more) systems with each of the block triangular blocks $\tilde{D}$. Even at the innermost level this is balanced by not having to multiply by the filled border blocks $\tilde{D}^{-1}B$; the fill results in Tables 6.1 and 6.2 suggest that this fill and the cost of using it is considerable. In addition the dense diagonal blocks may may permit the efficient use of vector hardware on machines like the CRAY-1.

We have made preliminary tests on the P^4 and P^5 algorithms, using a collection of chemical process flow problems obtained from A. Westerberg, and using a small number of LP bases from the sparse matrix collection of Iain Duff and John Reid. We used the P^4 code of Bisschop, Levy and Meeraus [1], and modified this code to produce a P^5 code. Both codes were used to provide orderings only -- a modification of the Harwell MA28 code [4] was used to perform a sparse symbolic factorization and to allocate sparse data structures. In

addition the symbolic factorizer was used to determine the structure of $\hat{S}$ and the transversal assignment in MA28 was used to obtain a suitable ordering for $\hat{S}$. In addition, we used the Harwell code to provide two different Markowitz-like orderings. In one we used no pivoting for numerical stability which gives a pure Markowitz pre-ordering which is directly comparable to the P^4 and P^5 orderings. For the other we used the numerical pivoting options of MA28 with a pivot tolerance of 0.1 to provide a numerically stable ordering.

These orderings were used in conjunction with the Harwell MA28B subroutine to perform sparse Gaussian elimination without pivoting. The MA28 subroutines provide for a reducible factorization for a matrix in block triangular form and this capability was used for all orderings. This results in a factorization which is implicit at the outer block triangular level, but explicit within each diagonal block. Neither the P^4 nor the Markowitz ordering permit us to do any better than this, since the structure of the diagonal blocks does not allow use of the implicit factorization ideas. For the P^5 ordering we computed the fill in the Schur complement blocks $\hat{S}$ within each outer diagonal block, the sum of which is the fill for a reducible and implicit factorization. To demonstrate the power of the alternative factorizations with the P^5 ordering we have included the fill results for using the outer reducible factorization with an explicit factorization for the diagonal blocks; in addition, for the chemical process models in Table 6.1 we give the fill for an explicit LU factorization of the entire matrix.

The results of our testing are summarized in Tables 6.1 and 6.2. We expect that the fill from the P^5 ordering without using the implicit factorization to be greater than the fill from the P^4 ordering and that is borne out in general. However, the the major result is the effect of using the block structure of the P^5 ordering, which changes the picture entirely. The reducible and implicit factorization for the P^5 ordering is clearly superior to P^4 and is quite competitive with the pure Markowitz ordering. The other surprising result is the frequency with which the P^4 ordering demonstrates that it is not structurally stable by leaving explicit zero entries on the diagonal of the reordered matrix.

Any production implementation of the P^5 algorithm must take into account that structural stability by no means guarantees numerical stability. Even though zero pivots are avoided, small pivots can arise as with any sparse ordering for general problems which does not recognize numerical values. Any pivoting which is restricted to the innermost diagonal blocks will not disturb the block structure of the matrix as ordered by P^5. Thus partial or complete pivoting within a diagonal block of the bordered block triangular form can be used to improve numerical stability. Pivoting restricted to certain sets of rows or columns is not sufficient to guarantee numerical stability in general, but it may suffice for certain classes of problems. The results we obtained on our limited test set are not particularily encouraging -- typically the innermost diagonal blocks had very low order, usually one, which does not provide much opportunity for

pivoting. In addition we have encountered at least one real problem for which an inner diagonal block was numerically singular.

The only form of numerical pivoting which is known by the authors to be used with the P^4 algorithm uses partial pivoting across rows. Such pivoting results in the interchange or "swap" of columns in the border with assigned columns. This interferes with the heuristic's means of maintaining the sparseness of the factored matrix, to the extent that the P^4 algorithm is not useful on problems where extensive pivoting results ([16]). Such pivoting within the P^5 algorithm interferes with the reducible and implicit block factorization. It is possible that the combination of complete pivoting within diagonal blocks and threshhold pivoting between diagonal blocks and the border may result in fewer interchanges than would occur with P^4, but this would be at the cost of destroying the block form.

Completely reliable numerical factorizations with preservation of sparsity with the P^5 ordering may require more dramatic means. Two possibilities now under consideration are to develop a further extension of the Hellerman-Rarick algorithms to incorporate the sizes of the numerical entries in the heuristic or to extend sparse least squares algorithms to provide a (completely stable) LQ factorization of the bordered block triangular form. Any successes with these approaches will be reported later.

TABLE 6.1

COMPARISON OF FILL INDUCED BY MARKOWITZ, P^4 AND P^5 ALGORITHMS
ON CHEMICAL PROCESS MATRICES

Order	Number of original nonzeroes	FILL					
		Markowitz without stability pivoting	Markowitz with stability pivoting	P^4	P^5 reducible & implicit	P^5 reducible	P^5 explicit
67	294	255	267	343	134	484	484
132	414	79	104	107	18	107	158
156	371	26	28	35	13	42	53
167	507	77	98	107	18	107	158
381	2157	2084	2136	F	2489	7047	7117
479	1910	969	1243	F	654	2392	3863
497	1727	241	316	632	196	535	1149
655	2854	1948	2802	F	1591	5833	8763
989	3537	961	1245	F	1444	6294	12116
1505	5445	1727	2274	F	2740	10870	*

An "F" in the column for the fill from the P^4 algorithm indicates that the P^4 ordering yielded a structurally zero diagonal element which prevented completion of the symbolic factorization and determination of the amount of fill.

* - exceeded memory allocation of program

TABLE 6.2

COMPARISON OF FILL INDUCED BY MARKOWITZ, P^4 AND P^5 ALGORITHMS
ON LINEAR PROGRAMMING BASES

Problem & Order	Number of original nonzeroes	FILL				
		Markowitz without stability pivoting	Markowitz with stability pivoting	P^4	P^5 reducible & implicit	P^5 reducible
STAIR 0* 363	2454	0	0	0	0	0
STAIR200* 363	3068	440	493	F	377	1241
STAIR400 363	3157	467	547	F	227	896
STAIR600 363	3279	943	1196	F	910	3142
BP 0* 822	3276	0	0	0	0	0
BP 200 822	3802	102	106	139	19	143
BP 400 822	4028	264	265	F	144	566
BP 600 822	4172	452	494	F	232	1060
BP 800 822	4532	638	675	F	620	2238
BP 1000 822	4661	745	766	F	618	2256
BP 1200 822	4726	942	959	F	764	2577
BP 1400 822	4790	1280	1293	F	1263	4922
BP 1600 822	4841	829	852	F	193	1986

An "F" in the column for the fill from the P^4 algorithm indicates that the P^4 ordering yielded a structurally zero diagonal element which prevented completion of the symbolic factorization and determination of the amount of fill.

* - bases which are triangular or nearly irreducible; an explicit P^5 factorization would be similar to the reducible factorization for these problems, but significantly worse for all of the other bases.

REFERENCES

[1] J. Bisschop, Y. Levy and A. Meeraus, Programs for Structure Analysis of Sparse Matrices, Technical Report, Development Research Center, World Bank, Washington, D. C., 1979, submitted to ACM Transactions on Mathematical Software.

[2] I. S. Duff, On Permutations to Block Triangular Form, J. Inst. Maths Applics 19 (1977), pp. 339-342.

[3] I. S. Duff, A Survey of Sparse Matrix Research, Proceedings of the IEEE 65 (1977), pp. 500-535.

[4] I. S. Duff, MA28 - a Set of FORTRAN Subroutines for Sparse Unsymmetric Linear Equations, AERE - R8730, Her Majesty's Stationery Office, London, 1977.

[5] I. S. Duff and J. K. Reid, An Implementation of Tarjan's Algorithm for the Block Triangularization of a Matrix, ACM Transactions on Mathematical Software 4 (1978), pp. 137-147.

[6] I. S. Duff and J. K. Reid, Algorithm 529: Permutations to Block Triangular Form, ACM Transactions on Mathematical Software 4 (1978), pp. 189-192.

[7] J. A. George, On Block Elimination for Sparse Matrices, SIAM J. Numer. Anal. 11 (1974), pp. 452-455.

[8] E. Hellerman and D. C. Rarick, Reinversion with the Preassigned Pivot Procedure, Mathematical Programming 1 (1971), pp. 195-216.

[9] E. Hellerman and D. C. Rarick, The Partitioned Preassigned Pivot Procedure (P^4), Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, 1972, pp. 67-76.

[10] T. D. Lin and R. S. H. Mah, Hierarchical Partition -- A New Optimal Pivoting Algorithm, Mathematical Programming 12 (1977), pp. 260-278.

[11] T. D. Lin and R. S. H. Mah, A Sparse Computation System for Process Design and Simulation, Part I. Data Structures and Processing Techniques, AIChE Journal 24 (1978), pp. 830-839.

[12] R. S. H. Mah and T. D. Lin, A Sparse Computation System for Process Design and Simulation, Part II. A Performance Evaluation Based on the Simulation of a Natural Gas Liquefaction Process, AIChE Journal 24 (1978), pp. 839-848.

[13] H. M. Markowitz, The Elimination Form of the Inverse and its Application to Linear Programming, Management Science 3 (1957), pp. 255-269.

[14] B. A. Murtagh and M. A. Saunders, MINOS User's Guide, Technical Report 77-9, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California, 1977.

[15] M. A. Saunders, Product Form of the Cholesky Factorization for Large-Scale Linear Programming, STAN-CS-72-301, Computer Science Department, Stanford University, Stanford, California, 1972.

[16] M. A. Saunders, A Fast, Stable Implementation of the Simplex Method Using Bartels-Golub Updating, Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 213-226.

[17] R. E. Tarjan, Depth First Search and Linear Graph Algorithms, SIAM Journal on Computing 1 (1972), pp. 146-160.

[18] A. Westerberg, private communication.