MANAGEMENT SCIENCE
Vol. 18, No. 1, September, 1971
Printed in U.S.A.

REARRANGING MATRICES TO BLOCK-ANGULAR FORM FOR DECOMPOSITION (AND OTHER) ALGORITHMS*

ROMAN L. WEIL† AND PAUL C. KETTLER‡§

University of Chicago

The rows and columns of an arbitrary coefficient matrix of large numerical problems can often be permuted so that substantial time can be saved in computations. For example, if a large linear programming problem has a suitable block-angular structure, one of the time-saving decomposition algorithms can be used. This article presents a systematic method for effecting such a block-angular permutation. An example and the results of manipulations of matrices with more than 300 rows and 2500 columns are shown.

1. Introduction

The readers of this journal are perhaps most familiar with the exploitation of block-angular structures in the context of mathematical programming. For example, if the rows and columns of the coefficient matrix of a mathematical programming problem can be rearranged so that the matrix has block-angular form (see matrix (1) below), then the well-known time-saving decomposition algorithms of Dantzig and Wolfe [4], Rosen [14], and Bennett [3] can be used. For other numerical calculations the discovery and exploitation of block angularity can be just as useful as in programming. We describe here a method for finding block-angular structures in matrices. For the most part we shall write as though the context were linear programming, but the actual context is inessential to the description.

To be sure, if a linear program can be put into block-angular form, then the canonical structure of the problem is often obvious from the semantics of the problem. For example: individual oil fields are subproblems and the refinery is the main problem; Chevrolet, Pontiac, Fisher Body, etc., are subproblems and the General Motors Headquarters is the main problem. Beale, Hughes, and Small [2] show one example of such an obvious structure. We believe, however, that no matter how large a computer is available, there will always be linear programs that are too large to be solved by the usual variants of the simplex method. These problems, *if they are to be solved at all*, must be broken down into parts that can be solved individually. To use the Dantzig-Wolfe, Rosen, or Bennett decomposition algorithms, one must have a block-angular structure or a systematic method for finding it.

Further, in real applications of linear programming, there is often great scope for redefining either the variables or the constraints to affect the structure of the coefficient matrix. Such techniques, by their very nature, must be specific to a particular problem. We have not been able to generalize a useful theory of variable or constraint redefinition.

A linear program with m rows and n columns has "only" $m!n!$ possible rearrangements of its rows and columns but not all of them need be structurally different from

* Received February 1969.

† University of Chicago.

‡ University of California, Berkeley.

§ We thank Robert L. Graves and a referee who provided useful advice. The research was supported by the National Science Foundation through a grant to the University of Chicago.

one another.[1] In theory, then, one systematic technique for finding the canonical structure is complete enumeration. But we can do much better.

In the following sections we specify the problem, outline the algorithm for solving it while illustrating it with an example, specify the steps of the algorithm, and relate some of our computing experience. In another article, [13], we give some details of the computer implementation of the algorithm. Few of the techniques we use are original with us. Their combination is. We have borrowed freely from directed and bipartite graph theory, particularly the work of Dulmage and Mendelsohn [5], [6], Harary [10], [11], and Steward [15], [16], [17].

2. The Problem

When a coefficient matrix has the structure shown in (1), that matrix is said to be in *canonical block-angular* form.

$$\begin{bmatrix} A_{01} & A_{02} & \cdots & A_{0n} \\ A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & \cdots & A_{nn} \end{bmatrix}. \tag{1}$$

The A_{ij} are arbitrary rectangular matrices, and the zero blocks contain only zero coefficients. In a decomposed linear program, the first row of matrices defines the relations of the main program, and the blocks below the first row on the main diagonal define the subprograms. Our problem was to construct a systematic method of answering these questions:

(a) Given an arbitrary matrix A, is there a rearrangement of its rows and columns to effect the canonical structure (1)?

(b) What is that rearrangement?

(c) Given that some matrices can be put into the canonical structure in more than one way, what is the "best" such rearrangement?

We have succeeded in answering the first two questions, but have only partial answers to the third.

3. Outline of the Algorithm with an Example

In this section we illustrate the algorithm with a sample problem. We show the various stages of the rearranging process to motivate the tedious explanations of the actual steps and to provide a concrete example for the interested reader. The example in this section does not include all possible complications, but enough to illustrate the algorithm in action. The matrix in Figure 1 shows the essential structure of an A matrix for a problem with 26 rows and 38 columns. All nonzero elements of the A matrix are denoted by "1"; all other elements of the A matrix are zeros and are

[1] The essential information is whether a coefficient is zero or nonzero. A matrix with all nonzero elements has only one structure, no matter how the rows and columns are rearranged. Counting the number of essentially different rearrangements for a given matrix is hard, irrelevant to our problem, but solvable short of complete enumeration by the combinatorial techniques of Polya's counting theory. See, for example, Golomb [7].

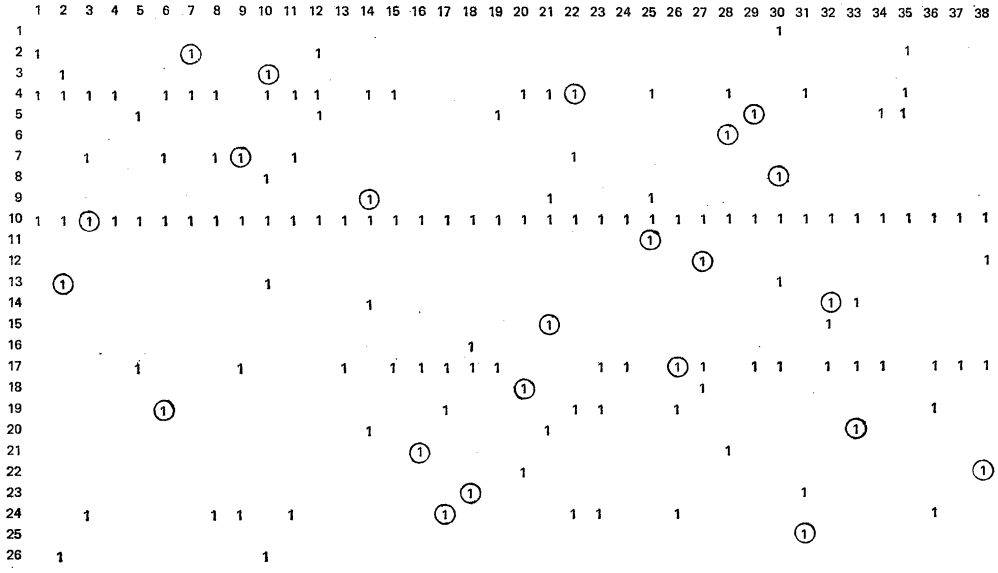

FIGURE 1. The original input matrix

represented by blanks in Figure 1. When we write "a row (or column) belongs to subprogram S", we mean that all of the nonzero elements of that row (or column) occur in subprogram S.

The first step in the algorithm is to mark (in Figure 1, elements are circled) the nonzero elements of A in such a way that the largest possible number of elements is marked, and no row nor column contains more than one mark. This procedure is formally called "finding a maximum transversal for the bipartite graph represented by A". In the example there are 23 marked elements. It is not possible to mark more than 23 elements in the required way, but there are other ways to mark 23 elements. Efficient means of finding the maximum transversal are presented below.

In Figure 2 the rows and columns are permuted so that the marked elements are brought to the main diagonal of the 23 by 23 square matrix, referred to as A^* below, in the northwest corner of A. Note that the southeast corner of A contains only zero elements: if there were nonzero elements in the southeast corner then the transversal would not be maximum.

In Figure 3 the matrix A^* is maximally decomposed. A square matrix D is decomposed (or reduced) when a rearrangement of its rows and columns is found so that the matrix has the form (2) in which the D_{ii} are square and all elements below the main

$$\begin{bmatrix} D_{11} & D_{12} & \cdots & D_{1m} \\ 0 & D_{22} & \cdots & D_{2m} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & D_{mm} \end{bmatrix} \tag{2}$$

block diagonal are zero. The elements above the main block diagonal may be nonzero. A matrix is maximally decomposed when m in (2) is as large as possible. The various ways to find the maximal decomposition are discussed below. If, in the maximal decomposition of A^*, m is one, then the matrix A is *indecomposable* and has no rear-

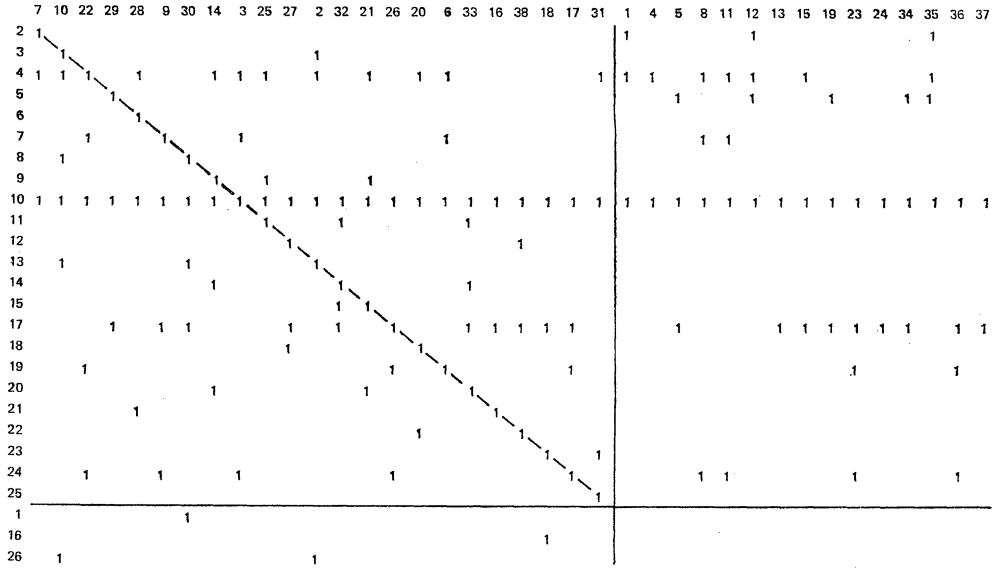

FIGURE 2. The matrix A^* exhibited in the northwest corner

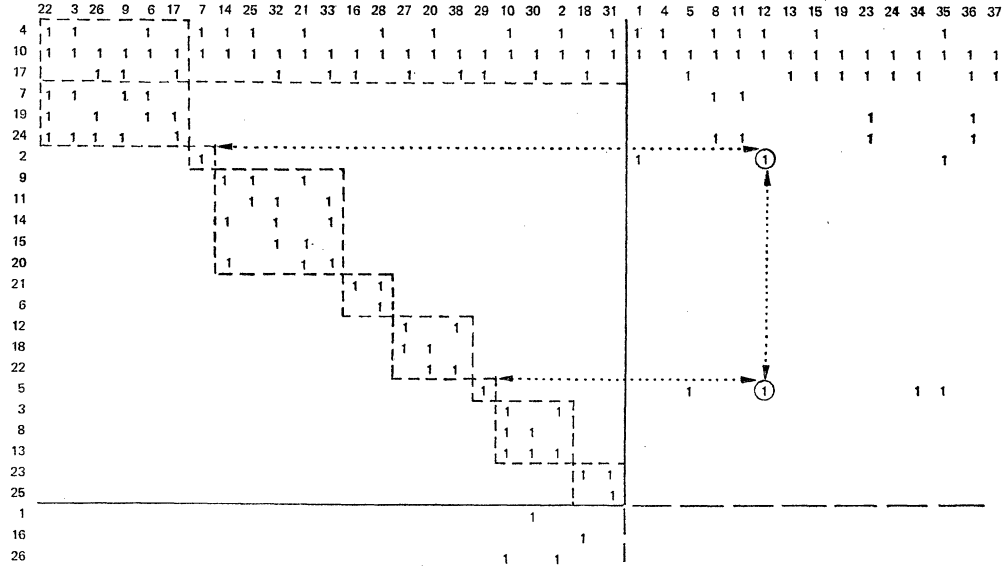

FIGURE 3. A^* decomposed in the northwest corner

rangement of its rows and columns to the canonical block-angular form. An m greater than one is necessary, but not sufficient to ensure the existence of the canonical structure.[2]

[2] To see how the example is less complicated than the most general matrix, note in Figure 3 that above the block diagonal, below the first three rows, all elements are zero. Matrices which have an eventual decomposition into canonical form need not be so sparse. There may be more than one way to construct a maximal transversal; there may be different A^* submatrices cor-

The full 26 × 38 matrix is still denoted "A" although rows and columns have been permuted since its original specification in Figure 1.

The canonical structure is beginning to appear: the first three rows of the matrix in Figure 3 are going to be in the main program (row zero) of the canonical structure. The diagonal blocks are going to be the subprograms of the canonical structure. So far, the columns to the right of A^* and the rows below A^* have been ignored. Now we must take these into account.

The column labeled "1" is the first as yet unassigned column. Note that below the main program rows, the only nonzero element in the column is in the row labeled "2". We assign column 1 to the subprogram including row 2, which until the assignment was the 1 × 1 subprogram of row 2 and column 7.

Next, column 4 is processed. Below the tentative main program rows column 4 has no nonzero elements. Thus column 4 need not be assigned to a subprogram, and we note to put column 4 on the far right of the canonical structure of A. Column 5 is processed next and assigned to the (formerly 1 × 1) subproblem of row 5 and column 29. Columns 8 and 11, the next two to be processed, are assigned to the (formerly 3 × 5) subproblem of rows 7, 19, 24, and columns 22, 3, 26, 9, 6, 17.

Column 12 presents a new complication. Its nonzero elements below the main problem occur in rows of two different subprograms—those of row 2, column 7 and row 5, column 29. Combine into a single subprogram the as yet unassigned column and the two or more subprograms "linked" by that column.

All of the columns of A not in A^* are processed in one of these ways:

(a) assigned to a subprogram straightaway,

(b) assigned to no subprogram and put on the right of A, adding a variable to the main program only, or

(c) assigned to more than one subprogram, connecting these subproblems into one.

Next the rows of A not in A^* are processed. If all the nonzero elements of such a row are in columns of a single subprogram, the row may be assigned to that subprogram. If a row has nonzero elements in columns of two or more subprograms, the row and the subprograms including the columns containing the nonzero row elements may be assigned to a single new subprogram. Or a row may be assigned to the main program. In Figure 3 of the example, rows 1 and 26 can be assigned to the next-to-last subprogram and row 16 to the last subprogram.

The rows and columns of A are permuted taking into account the assignments and connections just made. The result of these permutations is the canonical block-angular structure shown in Figure 4.

Note, however, that we may not yet have the canonical structure. It need not yet be true that all elements below the main program rows above the subprogram blocks be zero. For example, if in the original input matrix row 8, column 18 were nonzero, at this step there would be a nonzero element outside the blocked rectangles. The simplest thing to do would be to move row 8 up into the main problem to establish the canonical structure. Alternatively, the last eight rows could be joined into one subproblem. There will, in general, be such arbitrary choices to make. The arbitrary choices can be made in a way to minimize the expected computation time for the particular application at hand. See the further discussion below.

---

responding to different transversals; thus m may depend on which of several possible maximal transversals is chosen. If, for example, in Figure 2, row 16 is put into A^* in place of row 23, with element (16, 18) replacing (23, 18) on the maximal transversal, then in Figure 3, the lower-right two-by-two matrix will become two one-by-one matrices so that m would be 9, not 8.

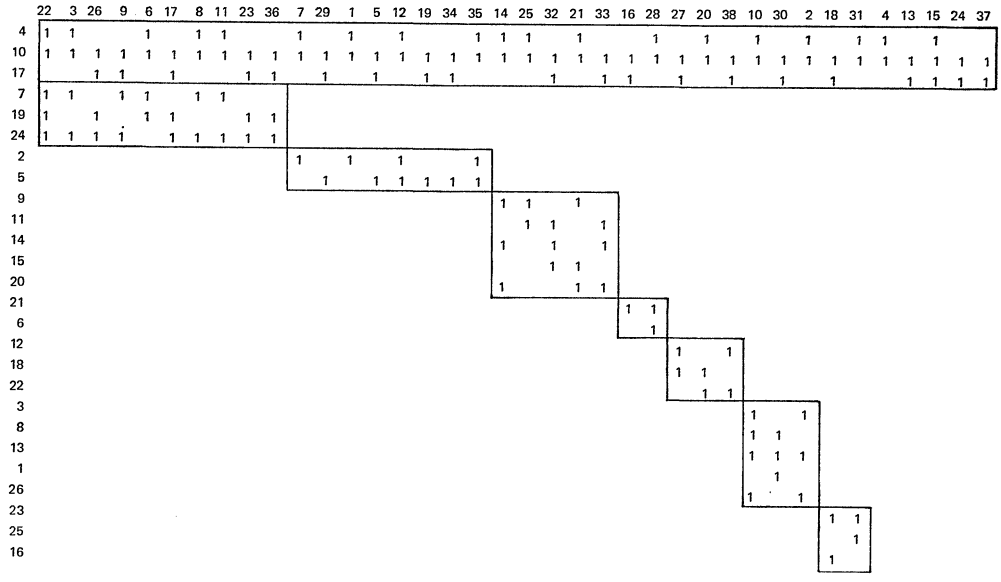

FIGURE 4. The canonical structure of A for the decomposition linear programming algorithm

4. The Steps of the Algorithm

Here we detail the three basic steps used to find the canonical block-angular structure. The problem and this section split conveniently into three subparts:

(a) The maximum transversal is identified and the matrix permuted so that the transversal occupies the first elements of the main diagonal,

(b) the square matrix A^* about the transversal main diagonal is maximally decomposed, and

(c) rows below the main program with coefficients in two or more subprograms are assigned to the main program, and rows and columns outside A^* are assigned to the subprograms or to the main program.

Identifying the Maximum Transversal

A maximum transversal in a matrix is a set of marked nonzero elements such that at most one mark appears in each row and in each column and such that the number of marked elements is as large as possible. We find the maximal transversal of a rectangular matrix of zeros and ones by interpreting that matrix as the incidence matrix of a bipartite graph[3] and by applying techniques developed by the graph theorists. Each row and column of the matrix represents a distinct node in the bipartite graph. The (i, j) element of the incidence matrix being 1 implies there is an arc connecting the node represented by row i and the node represented by column j. A maximal transversal in the incidence matrix corresponds to a largest set of arcs in the bipartite graph such that no node is the endpoint of more than one arc in the set.

[3] A bipartite graph is a graph in which all arcs connect nodes in disjoint sets. In Figure 4, the main problem (rows 4, 10, 17) and the first two (longer than tall) subproblems (rows 7, 19, 24, and rows 2, 5) constitute the *horizontal tail of the bipartite graph* represented by A. The next three (square) subproblems constitute the *irreducible subgraphs of the core* of the bipartite graph and the final two (taller than long) subproblems constitute the *vertical tail* of the bipartite graph. See [5], [6].

Identifying a maximum transversal in the incidence matrix of a bipartite graph is the same problem as finding the maximum *output set* in a system of equations (see Steward [15] or Weil and Steward [17]) or finding a system of distinct representatives (see P. Hall [9] or M. Hall [8]). Dulmage and Mendelsohn [5], [6] give an algorithm for finding a maximum transversal based on the algorithm in [8]. We have compared this algorithm with one adapted from Weil and Steward [17] for output sets. Of these two algorithms, only the Weil-Steward method will process a matrix in any configuration. The Dulmage-Mendelsohn method requires an initial matrix with a transversal that cannot be extended without displacing an element already in the transversal. Such a tableau can easily be provided by the iterative scheme known as the "northwest corner rule". This rule is in fact a good starting point for both algorithms.

Briefly, the northwest corner rule proceeds as follows:

(a) Rows of the matrix are successively scanned until a "1" is located.

(b) The appropriate rows and columns are then permuted to move the "1" to the northwest (i.e., upper left) corner of the matrix.

(c) The procedure is repeated on the rows and columns of the matrix not yet included in the transversal. These iterations produce a matrix with a string of 1's down the main diagonal. The procedure ends when there are no 1's in the southeast. Note that the transversal cannot then be extended without altering any of the current transversal elements.

Given the northwest corner start, both algorithms for finding a maximal transversal are operationally equivalent. They both set up chains of 1's to extend the transversal. A chain is a sequence of 1's such that two adjacent links appear in the same row or column of the matrix alternately. For example, the matrix entries $\{a_{11}; a_{13}; a_{43}; a_{45}; a_{25}\}$ constitute a chain if all are 1's. Chains are constructed such that: (i) Each chain contains an odd number of links, (ii) the first, last, and all other odd numbered links are not transversal elements, (iii) the second, next-to-last and all other even numbered links *are* transversal elements, (iv) the first and last links are *outside* the northwest corner. When such a chain can be constructed, the even elements are discarded from the transversal and the odd elements are inserted, necessarily increasing the length of the transversal by one. When no further chains with properties (i), (ii), (iii) and (iv) can be found, the current transversal is maximal. The details of such chain construction can be found in [5] or in [6].

Decomposing the Square Matrix

The rows and columns of the square matrix with the transversal on the principal diagonal must be maximally decomposed. A matrix is decomposable when its rows and columns can be permuted so that the matrix has the form of (2).[4]

People have been interested in decomposing matrices for some time.[5] Harary [10],

[4] Such a matrix is sometimes called *reducible*. The terminology on decomposable matrices has changed somewhat in recent years. Ando, Fisher, and Simon [1] have suggested a taxonomy that is becoming standard. If in matrix (2) the off-diagonal blocks are all zero (not just the ones below the main diagonal), the matrix is called *completely decomposable*.

[5] There are really two kinds of decomposable matrices that are not well enough differentiated. For both kinds rows and columns are permuted to a form like (2) but for some applications, for example, finding the eigenvalues of the matrix, the permutations of the rows and columns must be identical or simultaneous. Let P_1 and P_2 be permutation matrices (identity matrices with rows and columns permuted). Then $A^* = P_1AP_2$ is a way to define a permutation of the rows and columns of A. For a *simultaneous* permutation P_2 must equal the transpose or inverse of P_1. In this paper, there need be no such relation between P_1 and P_2—permutations need not be simultaneous.

[11] is the first to reference and solve the problem in the literature we have studied. Isolating ergodic sets of states in Markov chains is a problem in decomposing a square matrix, but we cannot find early literature references to algorithms for such techniques.

Almost any text on graph theory, e.g., Harary, Norman, and Cartwright [12, Chapter 5], will show the "power method" of decomposing square incidence methods. This method involves an analysis of the nth power, called the reachability matrix, of the $n \times n$ incidence matrix. Of the more modern techniques, we found the technique of Steward [15] to be the fastest. To describe the algorithm for maximal decomposition would be to repeat information which is easily available in the published literature.

The decomposition of the matrix A^* is the costly step in the whole algorithm so that attention to efficiency of decomposition is warranted. If A^* is not decomposable, the original matrix A can have no block-angular canonical structure. Each of the diagonal blocks below the first is a potential subprogram so that the number of blocks is a practical[6] upper limit to the number of subprograms. The off-diagonal elements of the decomposition can link subprograms. An $a_{ij} \neq 0$ not in a diagonal block potentially links the subprograms in which row i appears to the subprogram in which column j appears. The row in which such an a_{ij} occurs can be moved to the main problem so that the subprograms remain independent.

Final Manipulation of Rows and Columns

Rows and columns outside the matrix A^* are brought into the tentative main program and subprograms induced by the maximal decomposition. The process may require that two or more subprograms be combined into a larger subprogram.

First, identify the rows of A^* which will constitute the main program. These are the rows for which there is a nonzero element outside the diagonal of blocks. Each row below A^* may be moved into the main program or it may be put into the subprograms. To put such a row into the subprograms, scan it for nonzero elements. The columns containing these nonzero elements are marked. The subprograms containing the marked columns are combined along with the just-scanned row into a single subprogram.

Then columns to the right of A^* are assigned to subprograms. If such a column j has 1's in rows $i_1, i_2, \cdots, i_k$, where $i_1, i_2, \cdots, i_k$ are not in the main program, then the subprograms containing rows $i_1, \cdots, i_k$ are combined into a single subprogram to which column j is assigned.[7]

To resolve ambiguity that may arise when attempting to assign rows to the main program or to a group of subprograms, one needs a criterion function for evaluating the restructured matrix in its canonical form. The criterion should depend upon the computations that will be performed on the restructured matrix.

For most implementations of the decomposition algorithm for linear programs, a sensible criterion for a matrix with k subproblems would be to minimize

$$\sum_{i=1}^{k}(n_p + n_{s_i})^2,$$

[6] In the next subsection it will be made clear why we say that the number of blocks in the maximally decomposed version of A^* is a "practical" rather than theoretical upper bound to the number of blocks in the canonical block-angular form.

[7] Note, however, in the mathematical programming context that the variable represented by a single column that links two otherwise independent subproblems can always be replaced by two variables representing the contributions of the original variable to the separate subproblems provided that an additional constraint is appended to the main problem. The new constraint would require the two new variables to be equal.

where n_p is the minimum of the number of rows or number of columns in the main problem and n_{s_i} is the minimum of the number of rows or columns in subproblem i.

Observe in Figure 4 that row 23 could be put into the main problem so that rows 16 and 25 each become separate subproblems. To make such a shift would not improve the resulting canonical matrix measured by the criteria mentioned above, nor can we imagine an implementation for which it would. Whatever decomposition implementation is used, it appears safe to assert that, when given a choice, one prefers a row put into a subprogram, even if it ties two previously independent subprograms together, rather than into the main program.[8]

5. Applications to Large Matrices and Computing Experience

If the data for a problem can fit into the core of a computer along with the program to solve the problem, then the algorithm to find block-angular structure is of no interest. Therefore, if the implementation of our technique for finding the canonical structure of a coefficient matrix uses one computer word for every matrix entry, then the technique is of no interest. That is, if the problem is small enough to fit into core for our technique on a one-coefficient, one-word basis, then the problem will likely fit into core while the problem is being solved.

Our technique is only of practical use when it can process a much larger problem than can fit into core on a one-coefficient, one-word basis. There are at least two ways to use our technique for problems larger than will fit into core. The first is to allocate two items of information for every nonzero coefficient in the linear program coefficient matrix: the label of the row and the label of the column in which the nonzero element appears. These two items can be packed into one computer word so that, in total, as many words will be required as there are nonzero elements in the coefficient matrix. We discuss our program to implement this method in [13]. Compared to the requirements of the second method, the machine-language program for processing data stored in this form is more complex and execution time is slower. Under certain conditions this first method can handle larger problems.

The second method is to allocate one bit of core to each coefficient matrix entry. The bit is zero if the coefficient is zero, and one if not. The program to find block-angularity is relatively straightforward and execution is extremely fast. Further, it is easier to write about and understand this method; we have described our algorithm in the sections above as though this second method were the implementation chosen. Details are in [13].

On the IBM 7094 problems of size 50 by 70 were solved in only a few seconds by the second method. Table 1 summarizes computing experience on a sample of relatively small matrices. Note that execution time is an inverted U-shaped function of coefficient density. Of much more interest was the problem of 330 rows and 2560 columns that was solved in less than five and one-half minutes, including assembly and peripheral activity. We have reason to believe from study of other problems that even the most complex problem of this size would take no more than fifteen minutes. The time used to process the large matrix was broken down approximately as follows: 1.50 minutes for assembly card reading, and matrix packing, 3.90 minutes for processing, and about .10 minutes for printout.

[8] One referee does not believe this to be a safe assertion. Still, we cannot think of examples that render it unsafe. At any rate, the statement in Footnote 6 is now justified. The number of blocks in the maximally decomposed matrix A^* is not a theoretical upper limit to the number of subprograms.

TABLE 1

Average Internal (7094) Time in Seconds for Finding Block-Angular Form for Randomly Generated Matrices (10 Matrices Per Table Entry) Stored One Bit Per Coefficient

Matrix Size	Density: Percentage of Nonzero Elements			
	2	4	6	8
16 × 20	.19	.26	.31	.29
32 × 40	1.17	1.57	1.71	1.69
48 × 60	3.73	4.30	3.94	3.45
64 × 80	7.72	8.33	6.93	5.43

The largest problem we can solve directly on the 7094 with the second method is for a matrix with about 900,000 entries—say 300 rows and 3000 columns. One *bit* of core is used for each coefficient of the linear programming matrix, but two *words* in addition are needed for each column of the matrix.[9] In one application that was too large for our program by only several rows, we arbitrarily assigned to the main problem the rows with the most entries in order to cut the problem down to solvable size. This heuristic need not always produce that canonical form which minimizes the objective function. At greater preprocessing cost, one can eliminate rows which have the same configuration as other rows to guarantee that the canonical structure is not altered while making the problem smaller. In short, there are a number of techniques that can be used to supplement the main algorithm in order to increase marginally the size of the solvable problem. We were not overly concerned with such techniques and shall not be until actual problems arise.

Our implementation [13] on a one-coefficient, one-bit basis allows us to process problems on the order of 36 times as many coefficients as would fit directly into core for crude codes which allocate one word for every coefficient. In comparison with the more sophisticated code which stores only nonzero coefficients and the information about where the coefficient appears in the matrix, we can only say that one procedure will manipulate a larger problem, but how much larger depends upon density.

References

1. ANDO, A., FISHER, F. M. AND SIMON, H. A., *Essays on the Structure of Social Science Models*, The M.I.T. Press, Cambridge, Massachusetts, 1963.
2. BEALE, E. M. L., HUGHES, P. A. B. AND SMALL, R. E., "Experiences in Using a Decomposition Program," *The Computer Journal*, Vol. 8 (April 1965), pp. 13–18.
3. BENNETT, J. M., "An Approach to Some Structured Linear Programming Problems," *Operations Research*, Vol. 14 (1966), pp. 636–645.
4. DANTZIG, G. B. AND WOLFE, P., "Decomposition Algorithm for Linear Programs," *Operations Research*, Vol. 8 (January 1960), pp. 101–111.
5. DULMAGE, A. L. AND MENDELSOHN, N. S., "Graphs and Matrices," in *Graph Theory and Theoretical Physics*, F. Harary, Ed., Academic Press, New York, 1967.
6. ——, "Two Algorithms for Bipartite Graphs," *SIAM Journal*, Vol. 11 (March 1963), pp. 183–194.
7. GOLOMB, S. W., "A Mathematical Theory of Discrete Classification," in *Fourth London Symposium on Information Theory*, C. Cherry, Ed., Butterworths, London, 1961.
8. HALL, M., JR., "An Algorithm for Distinct Representatives," *American Mathematical Monthly*, Vol. 58 (1956), pp. 716–717.

[9] Actually the bookkeeping requirements are based on max (number of rows, number of columns), but for most linear programming problems, the number of columns is larger than the number of rows.

9. HALL, P., "On Representatives of Subsets," *Journal of the London Mathematical Society*, Vol. 10 (1935), pp. 26–30.
10. HARARY, F., "A Graph Theoretic Method for the Complete Reduction of a Matrix with a View Toward Finding Its Eigenvalues," *Journal of Mathematics and Physics*, Vol. 38 (1959/60), pp. 104–111.
11. ——, "A Graph Theoretic Approach to Matrix Inversion by Partitioning," *Numerische Mathematik*, Vol. 4 (1962), pp. 128–135.
12. ——, NORMAN, R. AND CARTWRIGHT, D., *Structural Models—An Introduction to the Theory of Directed Graphs*, John Wiley and Sons, New York, 1965.
13. KETTLER, P. C. AND WEIL, R. L., "An Algorithm to Provide Structure for Decomposition," in *Sparse Matrix Proceedings*, R. A. Willoughby, Ed., I.B.M. Corp., Yorktown Heights, 1969, pp. 11–24.
14. ROSEN, J. B., "Primal Partition Programming for Block Diagonal Matrices," *Numerische Mathematik*, Vol. 6 (1964), pp. 250–260.
15. STEWARD, DONALD V., "On an Approach to Techniques for the Analysis of the Structure of Large Systems of Equations," *SIAM Review*, Vol. 4, No. 4 (October 1962), pp. 321–342.
16. ——, "Partitioning and Tearing Large Systems of Equations," *SIAM Journal, Series B*, Vol. 2, No. 2 (1965), pp. 345–365.
17. WEIL, R. L. AND STEWARD, D. V., "The Question of Determinacy in Square Systems of Equations," *Zeitschrift für Nationalökonomie*, Vol. 27 (1967) pp. 261–266.