Computing the Block Triangular Form of a Sparse Matrix

ALEX POTHEN and CHIN-JU FAN
The Pennsylvania State University

We consider the problem of permuting the rows and columns of a rectangular or square, unsymmetric sparse matrix to compute its block triangular form. This block triangular form is based on a canonical decomposition of bipartite graphs induced by a maximum matching and was discovered by Dulmage and Mendelsohn. We describe implementations of algorithms to compute the block triangular form and provide computational results on sparse matrices from test collections. Several applications of the block triangular form are also included.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*sparse and very large systems*; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis*

General Terms: Algorithms

Additional Keywords and Phrases: Block triangular form, Dulmage–Mendelsohn decomposition, maximum matchings, sparse matrices

1. INTRODUCTION

We consider the problem of permuting the rows and columns of a sparse matrix with arbitrary row and column dimensions to compute its block triangular form (btf). Block triangularization of a sparse matrix leads to savings in computational work and intermediate storage for many sparse matrix algorithms, including algorithms for solving linear systems of equations, the linear least squares problem, the null space problem, partitioning sparse matrices in parallel computation, and so forth. Inasmuch as block triangularization is equivalent to computing a particular decomposition of bipartite graphs, it has applications to problems outside the sparse matrix domain as well.

Algorithms for computing the btf of a sparse matrix are based on a canonical decomposition of bipartite graphs discovered by Dulmage and Mendelsohn. These algorithms rely on the concept of *matchings* in bipartite graphs, or equivalently,

The work of A. Pothen was supported by NSF grant CCR-8701723 and by U.S. Air Force Office of Scientific Research Grant AFOSR-88-0161.

Authors' addresses: A. Pothen, Department of Computer Science, The Pennsylvania State University, University Park, PA 16802; e-mail address: pothen@cs.psu.edu or na.pothen@na-net.stanford.edu; C.-J. Fan, Department of Computer Science, The Pennsylvania State University, University Park, PA 16802; e-mail address: fan@cs.psu.edu

on the dual concept of *vertex covers*. Sparse matrix researchers have studied a version of this decomposition applicable to square, unsymmetric, structurally nonsingular matrices, *inter alios* Duff and Reid [8], Erisman et al. [13], Gustavson [20], and Howell [22]. An implementation of an algorithm for computing the btf of such matrices is described in Duff and Reid [8], and their program MC13D is included in the Harwell subroutine library.

A more general btf exists for rectangular matrices and square, unsymmetric matrices that are structurally singular. This btf is based on a more general version of the Dulmage–Mendelsohn decomposition. The relationship between this more general btf and the btf of square, structurally nonsingular matrices will become clear in the next section. In this paper we report on the implementation of the algorithms that compute this more general btf and provide computational results on several sparse, rectangular matrices from different application areas.

The organization of our paper is as follows. In Section 2, we describe the Dulmage–Mendelsohn decomposition on which our program is based. Next, in Section 3, we describe our implementations of the algorithms that are used to compute the btf. These include a maximum matching algorithm and algorithms that compute the btf in two stages, which we call the *coarse decomposition* and the *fine decomposition*. In Section 4, we compute the block triangular forms for several sparse matrices from the Boeing–Harwell test collection and from the lp/data collection in the Netlib electronic software library. In the final section, we describe several applications in which the btf of sparse matrices plays a role.

2. THE DULMAGE–MENDELSOHN DECOMPOSITION

In this section we describe the btf of an $m \times n$ matrix A, where we assume for convenience that $m \geq n$. This assumption causes no loss of generality, for if the matrix is underdetermined, we can consider its transpose. It is helpful in the computation of the btf to consider the bipartite graph associated with A, $G(A) = (R, C, E)$. Here R is the set of vertices corresponding to the rows of A, C is the set of vertices corresponding to columns of A, and E is the set of edges corresponding to the nonzeros in A. Hence, R consists of m vertices, which we number from r_1 to r_m, and C consists of n vertices, numbered c_1 to c_n. For every nonzero entry a_{ij} in the matrix A, there is an edge (r_i, c_j) in E.

A *matching* M in $G(A)$ is a subset of its edges with no common endpoints. In the matrix A, this corresponds to a subset of nonzeros, no two of which belong to the same row or column. A node is *matched* if an edge in a matching is incident on it; otherwise it is *unmatched*. A bipartite graph $G(A)$ with a matching is shown in Figure 1, where the matched edges are indicated by dark horizontal lines. The matched nonzeros in the corresponding matrix A are indicated by the '⊗' symbol in Figure 2. Elements marked '×' are the other nonzeros in the matrix, and zero elements are indicated by empty entries.

A *walk* is a sequence of vertices $v_0, v_1, \ldots, v_{n-1}; v_n$ such that (v_i, v_{i+1}) is an edge for $i = 0, \ldots, n - 1$. Edges or vertices can be repeated in a walk. An *alternating walk* is a walk with alternate edges in a matching M. An *alternating tour* is an alternating walk whose endpoints are the same. An *alternating path* is an alternating walk with no repeated vertices. An *augmenting path* is an alternating path that begins and ends with unmatched nodes.

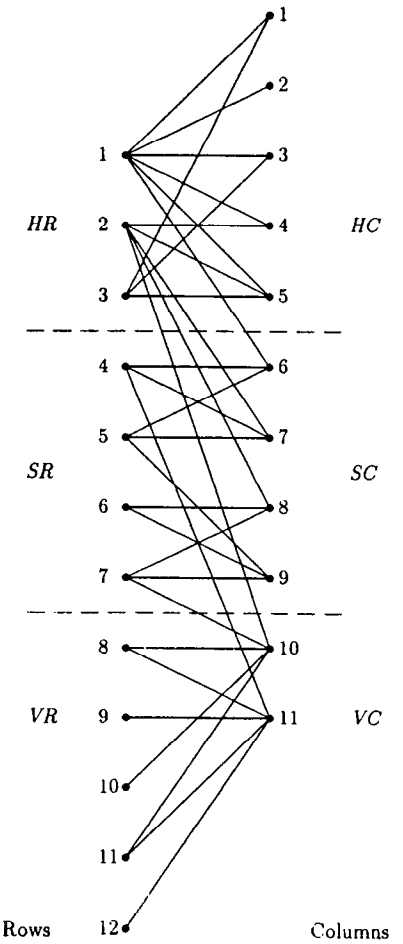


Fig. 1. The Dulmage–Mendelsohn decomposition of a bipartite graph.

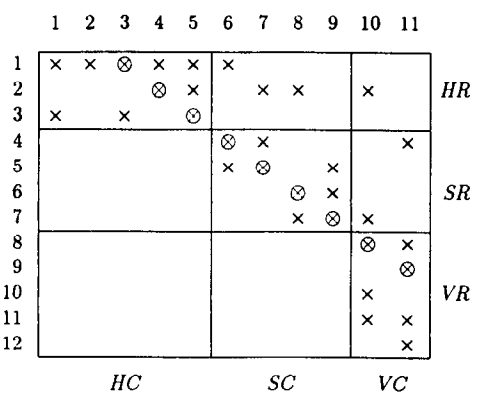


Fig. 2. The block triangular form of A.

The cardinality of a matching is the number of edges in it. An augmenting path can be used to increase the cardinality of a matching by interchanging its matched and unmatched edges. A *maximum matching* is a matching of maximum cardinality. This corresponds in the matrix to a diagonal with the maximum number of nonzeros in it. Berge [28] proved that a matching in a graph is maximum if and only if the graph contains no augmenting path with respect to it. A matching is *column-perfect* if every column vertex in C is matched; it is *row-perfect* if every row vertex in R is matched. A matching is *perfect* if it is column-perfect and row-perfect; this implies that R and C have equal sizes. The matching in Figures 1 and 2 is a maximum matching.

Lawler [25], Lovasz and Plummer [28], and Papadimitriou and Steiglitz [31] contain good discussions of matching theory and algorithms.

The $m \times n$ matrix A (with $m \geq n$) has the *Hall Property* (HP) if every subset of k columns has nonzeros in at least as many rows. Philip Hall [28] proved that A has a maximum matching in which all its columns are matched if and only if it has the Hall Property. A stronger requirement on A is the *Strong Hall Property* (SHP): every subset of $0 < k < m$ columns has nonzeros in at least $k + 1$ rows. (Thus, when $n < m$, every subset of $k \leq n$ columns has the required property, and when $n = m$, every subset of $k < n$ columns has the property.) The importance of the Strong Hall Property will become clear after the description of the Dulmage–Mendelsohn decomposition. Both these terms are due to Coleman, Gilbert, and Edenbrandt [3].

The Dulmage–Mendelsohn decomposition was described in a series of papers by Dulmage, Johnson, and Mendelsohn [10–12, 23]. We state the decomposition by a series of lemmas. Let M be a maximum matching in the bipartite graph of A, with row set R and column set C. With respect to M, we can define the following sets:

VR = {row vertices reachable by alternating path from some unmatched row}
HR = {row vertices reachable by alternating path from some unmatched column}
$SR = R \backslash (VR \cup HR)$
VC = {column vertices reachable by alternating path from some unmatched row}
HC = {column vertices reachable by alternating path from some unmatched column}
$SC = C \backslash (VC \cup HC)$.

The reader will find it helpful to consider the example in Figure 1 and its btf in Figure 2. In the btf, the rows and columns of A are permuted such that it has the block upper triangular form:

$$A = \begin{pmatrix} A_h & X & X \\ & A_s & X \\ & & A_v \end{pmatrix},$$

where A_h is underdetermined, A_s is square, A_v is overdetermined, and Xs denote possibly nonzero matrices of appropriate dimensions. (The prefix H stands for "horizontal", S for "square", and V for "vertical.")

The proofs of the following results may be found in [32].

LEMMA 2.1 *The sets VR, SR, and HR are pairwise disjoint; similarly the sets VC, SC, and HC are pairwise disjoint.*

LEMMA 2.2 *A matching edge joins a row vertex in VR only to a column vertex in VC; a row vertex in SR only to a column vertex in SC; and a row vertex in HR only to a column vertex in HC.*

LEMMA 2.3 *Row vertices in SR are perfectly matched to column vertices in SC.*

LEMMA 2.4 *No edge joins: a column vertex in HC to row vertices in SR or VR; a column vertex in SC to row vertices in VR.*

From the previous two lemmas, and from the construction of the various row and column sets, it follows that $|VR| > |VC|$, $|SR| = |SC|$, and $|HR| < |HC|$. Let us denote by (HR, HC) the submatrix of A induced by the row set HR and the column set HC, and similarly for the other row and column subsets. We call the submatrix $A_h = (HR, HC)$ the *horizontal submatrix*, the submatrix $A_s = (SR, SC)$ the *square submatrix*, and $A_v = (VR, VC)$ the *vertical submatrix*. The corresponding bipartite subgraphs are denoted G_h, G_s, G_v, respectively. The submatrix A_h is underdetermined, A_s is square, and A_v is overdetermined, as stated earlier. In addition, A_h has a row-perfect matching, A_s has a perfect matching, and A_v has a column-perfect matching. The above lemmas imply that A can be permuted into a block upper triangular form with diagonal blocks A_h, A_s, and A_v as shown in Figure 2.

LEMMA 2.5 *The submatrix A_v has the Strong Hall Property, as does the submatrix A_h^T.*

The importance of the SHP of these blocks in two sparse matrix problems, the null space problem, and the linear least squares problem, is described in Section 5.

The next theorem states that even though the Dulmage–Mendelsohn decomposition was stated with respect to a particular maximum matching, any other choice of a maximum matching would partition A into the same submatrices A_h, A_s, and A_v. Hence the vertical, square, and horizontal submatrices in the btf of A are unique.

THEOREM 2.1 *The sets VR, SR, HR and VC, SC, and HC are independent of the choice of the maximum matching M; hence the Dulmage–Mendelsohn decomposition is a canonical decomposition of the bipartite graph G.*

We call the above decomposition of A into the submatrices A_h, A_s, and A_v the *coarse decomposition*. One or two of the three submatrices may be absent in the coarse decomposition of a given matrix A. If A is overdetermined with a column-perfect matching, then A_h will be absent in its coarse decomposition; if A does not have a column-perfect matching, then A_h will be present. The submatrix A_s will be present or absent depending on the nonzero structure of A. Similarly, if A is underdetermined, the presence of the submatrix A_v will depend on whether it has a row-perfect matching or not. The presence of A_s will again depend on the structure of A.

If A is square and unsymmetric, there are two possible cases, depending on whether A has a perfect matching or not. (This corresponds to A being structurally nonsingular or singular.) If A has a perfect matching, then its coarse decomposition has only the submatrix A_s; otherwise, both A_h and A_v will be present.

It may be possible to further decompose the submatrices A_h, A_s, and A_v to obtain the *fine decomposition* of these submatrices. Each of A_h or A_v may be decomposable into block diagonal form; this corresponds to finding the connected components of G_h and G_v. If there are p connected components in G_h, then A_h has p diagonal blocks and can be permuted to the structure

$$A_h = \begin{pmatrix} A_{h1} & & & \\ & A_{h2} & & \\ & & \ddots & \\ & & & A_{hp} \end{pmatrix}.$$

Here, each diagonal block A_{hi} is underdetermined and has all of its rows matched to a subset of its columns.

The fine decomposition corresponding to A_v is similar to that of A_h, the only difference being that the diagonal blocks are now overdetermined.

The square submatrix A_s has a more interesting fine decomposition; it may have the block triangular form described below.

Consider the perfectly matched square submatrix A_s and the associated subgraph G_s induced by SR and SC. We call two-column vertices in SC *equivalent* if they lie on an alternating tour. This is an equivalence relation. Let the classes of this equivalence relation be $C_1, C_2, \ldots, C_p$, and let R_i be the set of rows matched to C_i.

LEMMA 2.6 *The row subsets $\{R_i\}$ and column subsets $\{C_i\}$ can be renumbered such that if C_i has nonzeros in the row set R_j, then $j \leq i$.*

We call this the *fine decomposition* of G_s. This decomposition for a graph G_s and the corresponding block upper triangular form for its matrix A_s are shown in Figure 3. (This graph and matrix are different from G_s, A_s in Figures 1 and 2.)

LEMMA 2.7 *The submatrix induced by R_i and C_i has SHP.*

THEOREM 2.2 *The partitions $R_1, \ldots, R_p$ and $C_1, \ldots, C_p$ are independent of the choice of maximum matching.*

THEOREM 2.3 *Let nonmatching edges in G_s be directed from columns to rows, matching edges shrunk into single vertices, and the vertices identified with the rows. The resulting directed graph G_d has $R_1, \ldots, R_p$ as its strongly connected components.*

This result enables us to use Tarjan's algorithm [34] for finding strongly connected components in a directed graph in order to find the block upper triangular form of the square submatrix. As noted above, if A is square, unsymmetric, and has a perfect matching, its coarse decomposition consists of the submatrix A_s alone. It is the fine decomposition of A_s that Duff and Reid [8] have implemented in their program MC13D. If A is square, unsymmetric, and

does not have a perfect matching, then MC13D puts the maximum number of nonzeros on the diagonal, treats the remaining zeros on the diagonal as nonzeros, and then computes a block upper triangular form corresponding to this diagonal. However, in this case, the matrix has a finer btf that can be obtained from the coarse and fine decompositions described here.

3. ALGORITHMS

We describe our algorithm to compute a Dulmage–Mendelsohn decomposition of an $m \times n$ matrix A, where, without loss of generality, we have assumed $m \geq n$.

We represent the nonzero structure of the matrix A by both column-oriented and row-oriented adjacency lists and pointer arrays used by SPARSPAK [15]. Thus the array *adjcol*(.) and the pointer array *xadjc*(.) are used to represent the column indices of nonzeros in rows of A. Similarly, arrays *adjrow*(.) and *xadjr*(.) are used to represent row indices of nonzeros in columns of A.

The algorithm to compute the btf has three phases:

Phase 1 Find a maximum matching M in the bipartite graph $G(A)$.

Phase 2 With respect to M, partition R into the sets VR, SR, HR; similarly partition C into the sets VC, SC, HC.

Phase 3 Find the diagonal blocks of the submatrix $A_v = (VR, VC)$, from the connected components of G_v; similarly for $A_h = (HR, HC)$.
Find the block upper triangular form of the submatrix $A_s = (SR, SC)$ by finding strong components in the associated directed graph G_d.

3.1 The Maximum Matching

Duff [5] has described the implementation of an $O(n\tau)$ maximum matching algorithm. More recently, Duff and Wiberg [9] have implemented an $O(\sqrt{n}\tau)$ algorithm due to Hopcroft and Karp. Either of these algorithms could be used to find the maximum matching in this phase. The latter algorithm has an asymptotically superior worst-case complexity. However, empirically, the running times of the algorithms are greatly improved by the use of heuristic features, and the relative performance of the two algorithms is problem dependent. Duff's implementation of the former algorithm is conceptually simpler and uses less storage. A detailed description of Duff's implementation of an $O(n\tau)$ matching algorithm may be found in the book by Duff et al. [6].

We implemented an algorithm similar to the one in [5], but found it to be slow on the denser problems we considered. This motivated the design of the variant of the $O(n\tau)$ maximum matching algorithm described below. We found that our implementation of this algorithm was competitive with Duff's $O(n\tau)$ algorithm on sparser problems, while it was faster than the latter on denser problems. A description of our $O(n\tau)$ matching algorithm follows.

Algorithm 1. Maximum Matching
Step 0. [Initialize]
Set the matching M and the set of unmatched columns U to be empty.
Step 1. [cheap matching]
for each column vertex $c \in C$ **do**
match c to the first unmatched row vertex $r \in adj(c)$, if there is such a row;
if c cannot be matched, add c to U;
end for

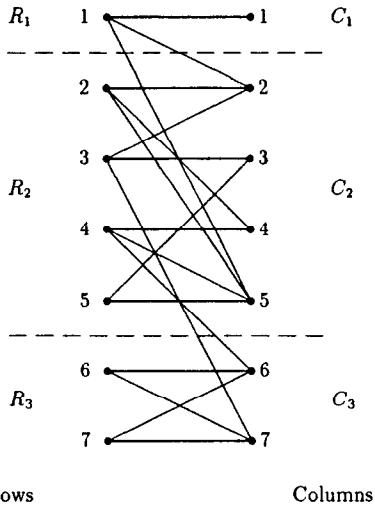
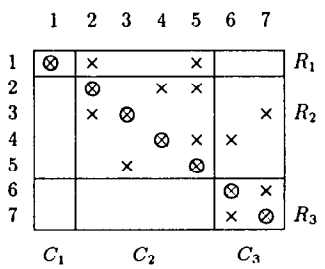


Fig. 3. The fine decomposition of G_s and the block upper triangular form of A_s.



Step 2. [augment matching]
$U_{new} := \emptyset$
repeat
{perform one pass of the augmenting procedure}
for each column vertex $c \in U$ **do**
search for an augmenting path from c,
visiting only row vertices that have not been visited previously in this pass;
mark all row vertices reached as visited;
if an augmenting path is found, augment M; else include c in U_{new};
end for
$U := U_{new}$; $U_{new} := \emptyset$;
until no augmenting path is found in a pass.

Inasmuch as an augmenting path has an unmatched column vertex at one end and an unmatched row vertex at the other, it is no loss of generality to search for augmenting paths from unmatched columns only. Since we are searching by alternating paths, the depth-first searches (dfs) have a simple structure. From each column vertex, we search all rows adjacent to it, but from a row vertex, we search only the column matched to it.

Duff et al. [6] call Step 1 finding a *cheap matching*. If A is a square, unsymmetric matrix of order n with a perfect matching, then, using the Hall property, it is

easy to show that at least $\lceil n/2 \rceil$ columns can be matched in the cheap-matching step. (This is Exercise 6.5 in [6].) We now prove that a similar result holds even when the matrix is overdetermined and does not have a column-perfect matching. We first prove the result using matrix theory, via the following theorem due to König, stated and proved in Minc [30] (Ch. 4, Theorem 2.2).

THEOREM 3.1 (König) *Let A be an $m \times n$ matrix, where $m \geq n$, and let $k \leq n$ be a nonnegative integer. A necessary and sufficient condition that the size of a maximum matching of A is $n - k$ is that it contains an $s \times t$ zero submatrix with $s + t = m + k$.*

LEMMA 3.1 *Let A be a matrix with dimensions as in Theorem* 3.1, *with a maximum matching of size $n - k$. The cheap-assignment phase of Algorithm* 1 *finds a matching of size at least $\lceil (n - k)/2 \rceil$.*

PROOF. Let l be the size of a matching obtained in the cheap-matching step. Reorder the rows and columns of A such that it can be partitioned as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},$$

where A_{11} is the $l \times l$ submatrix consisting of the matched rows and columns. The $(m - l) \times (n - l)$ submatrix A_{22} must be the zero submatrix, else the cheap phase would be able to extend the matching. Since A has a maximum matching of size $n - k$, by König's theorem,

$$(m - l) + (n - l) \leq m + k.$$

Simplifying, we get $l \geq (n - k)/2$. Since l is an integer, the lemma follows. □

An alternative proof of Lemma 3.1, which uses matching theory only, can be obtained as follows. Let L be a matching found by the cheap phase, M a maximum matching of size $n - k$, and N the empty matching. The symmetric difference $M \oplus N = (M \backslash N) \cup (N \backslash M)$ consists of the $(n - k)$ matched edges of M, forming a set of $n - k$ vertex disjoint-augmenting paths with respect to the empty matching N, each consisting of a single edge. Let (u, v) be any edge of the graph included in L. At most two edges of M may become ineligible to be included in L as a result of including (u, v), that is, an edge in M with u as one endpoint and another edge in M with v as one endpoint. Hence the matching L contains at least $\lceil (n - k)/2 \rceil$ edges.

We now turn to a consideration of Step 2 of Algorithm 1. This step is organized into *passes*; in each pass, we search for vertex-disjoint augmenting paths from all unmatched columns that are maintained in the set U. The cost of searching for the augmenting paths in a pass is at most $O(\tau)$, since the adjacency list of each column is examined at most once during the pass. When no augmenting path is found in a pass, the algorithm terminates. We will prove that then the algorithm does find a maximum matching.

THEOREM 3.2 *Algorithm* 1 *terminates with a maximum matching in the graph.*

PROOF. A matching is maximum if and only if there is no augmenting path in the graph with respect to it. Hence we prove that, at termination, there is no

augmenting path in the graph. Suppose, for a contradiction, that there exists an augmenting path in the graph when the algorithm terminates.

Number the unmatched columns in U in the last pass of the algorithm in the order in which the algorithm performs augmenting path searches from them. Let c_u be the smallest column in U such that there is an augmenting path in the graph beginning with this unmatched column. Let the corresponding augmenting path be

$$c_u, r_1, c_1, \ldots, r_{k-1}, c_{k-1}, r_k,$$

where (r_j, c_j) is a matched edge for $j = 1, \ldots, k-1$, and r_k is an umatched row. The algorithm failed to find this path when it performed a vertex disjoint dfs from c_u. This happened because some row on this path r_i, where $i < k$, was visited by an earlier dfs from an unmatched column c', during this pass. Then the alternating path from c' to r_i concatenated with the path from r_i to r_k is an augmenting path in the graph. This contradicts the choice of c_u. □

Step 1 of the algorithm can be implemented in $O(\tau)$ time. Each pass of the algorithm can also be implemented in $O(\tau)$ time. There can be at most $O(n)$ passes, since in each pass, except the last one, at least one more column is matched. Hence the complexity of the algorithm is $O(n\tau)$.

We have incorporated several features that have been previously employed by Duff to obtain an efficient implementation of Algorithm 1. We briefly describe them now. Detailed descriptions of these features may be found in Duff [5] and Duff et al. [6].

The depth-first searches are made efficient by a technique called *lookahead* (which is one step of a breadth-first search). Before performing the dfs from a column, all rows adjacent to it are examined to see if there is an unmatched row among them. If so, the dfs can be terminated, since an augmenting path from an unmatched column to an unmatched row has been found.

A pointer array into the adjacency lists of each column is maintained to ensure that each edge is examined at most once in Step 1, and the look-ahead part of the dfs in Step 2, over the entire algorithm. Hence the look-ahead feature costs only $O(\tau)$ over the whole algorithm. Another pointer array into the adjacency lists of the columns is used to ensure that each edge is examined at most once during the vertex-disjoint depth-first searches in a pass. Thus each pass can cost, at most, $O(\tau)$. An array is also used to mark rows that have already been visited during a dfs; the reinitialization of this array at the beginning of each pass can be avoided by using the number of the pass as a flag.

The matching is represented by means of two arrays *rowset* and *colset*. Row *ir* is matched to column *ic* if $rowset(ir) = ic$ and $colset(ic) = ir$.

It is appropriate at this point to compare Algorithm 1 to the Duff and Wiberg [9] implementation of the Hopcroft and Karp algorithm. The Hopcroft and Karp algorithm is organized into phases; in each phase, they find a maximal set of vertex disjoint shortest augmenting paths in the graph. This is accomplished by generating an auxiliary graph by breadth-first search (bfs) from unmatched rows and stopping the search at the level at which the first unmatched column is found. A maximal set of shortest augmenting paths is found by vertex disjoint depth-first searches from the unmatched columns (in the last level) of the

auxiliary graph. Each phase can be implemented in $O(\tau)$ time, and Hopcroft and Karp proved that there are at most $O(n^{1/2})$ phases in the algorithm.

Duff and Wiberg [9] found that in their implementation of the Hopcroft and Karp algorithm, the breadth-first searches dominated the running times and examined several ways of reducing this cost. They succeeded in finding a variant algorithm ((C4) in their paper), with the $O(n^{1/2}\tau)$ worst-case time bound, which was competitive with Duff's earlier implementation of the $O(n\tau)$ algorithm. In this variant algorithm, they found, during each phase, a maximal set of shortest augmenting paths in the auxiliary graph (generated by the bfs) by vertex-disjoint depth-first searches from the unmatched columns in the auxiliary graph. In addition, more augmenting paths were found by performing vertex-disjoint depth-first searches in the original graph from the remaining unmatched rows in that graph. These latter searches cost, at most $O(\tau)$ time and serve to reduce the total number of breadth-first searches in the algorithm. This variant algorithm performs well in practice compared to Duff's original $O(n\tau)$ algorithm. It is possible to implement this algorithm without explicitly generating and storing the auxiliary graph; only a few arrays for recording the level structure information and for maintaining the queues of unmatched rows and columns are needed.

Algorithm 1 finds a maximal set of vertex disjoint augmenting paths, without paying heed to the augmenting path lengths, by the vertex disjoint depth-first searches in each pass. Each pass may find several augmenting paths at the cost of $O(\tau)$ operations, but because shortest augmenting paths are not found, the worst-case complexity of the algorithm becomes $O(n\tau)$.

We compare the running times of Algorithm 1 implemented in Fortran 77 on a Sun 4/260 with implementations of Duff's $O(n\tau)$ algorithm and the Duff and Wiberg algorithm on two classes of problems in Table I. Our implementation of the latter two algorithms are variants of the implementations described in Duff [5] and Duff and Wiberg [9], modified to work with the Sparspak data structures that represent the matrix. Our implementation of the Duff and Wiberg algorithm requires extra storage only for two additional arrays of length m and two of length n, to mark the row and column level structures in the bfs and to maintain queues of unmatched row and column nodes. The programs were run on a Sun 4/260, and additional details about experimental conditions may be found in Section 4.

The first class of problems in Table I is a set of fairly dense (density $\approx$20 percent), rectangular (but nearly square) economy problems in the Boeing–Harwell test collection. On this class, Algorithm 1 and the Duff and Wiberg algorithm perform much better than Duff's algorithm. The second class of problems in the table is the set of Ncube problems (from Duff [5]), constructed to exhibit the worst-case $O(n\tau)$ time behavior of Duff's algorithm. It can be seen that Algorithm 1 has almost the same running times as Duff's algorithm, and hence also requires $O(n\tau)$ time on these examples, and that the Duff and Wiberg algorithm is much faster than both the former algorithms. In Section 4, Table IV, we compare the performance of Algorithm 1 with the Duff and Wiberg algorithm on a set of 28 linear programming constraint matrices from the netlib library. We find that Algorithm 1 is, on the average, more than three times faster than the Duff and Wiberg algorithm on this set of problems.

Table I. Time (in seconds) Required for Maximum Matching in Two Classes of Problems

Problem			Matching time		
Rows	Cols.	Nonzeros	Duff	Alg. 1	D-W Alg.
497	493	44551	5.03	0.36	0.33
497	493	53403	6.70	0.39	0.41
497	493	50409	7.07	0.38	0.38
492	490	41063	5.21	0.36	0.33
492	490	49920	6.51	0.38	0.38
492	490	49920	6.73	0.39	0.39
120	120	1760	0.12	0.14	0.03
240	240	6720	0.72	0.81	0.08
360	360	14880	2.28	2.57	0.15
480	480	26240	5.36	5.72	0.25
600	600	40800	10.12	11.02	0.37
660	660	49280	13.27	14.76	0.44
720	720	58560	17.42	18.88	0.58

Thus we find that the relative performance of Algorithm 1 and the Duff and Wiberg algorithm is problem-dependent. More experience with problems from a wide variety of application areas is needed before definitive conclusions can be drawn. Unfortunately, the number of rectangular test matrices in the Boeing–Harwell collection is fairly small, and this makes extensive testing difficult. However, the time taken by the matching algorithms on most problems is a tiny fraction of the time required for numerical factorization; for most linear programs in Table IV, it is also small in comparison with the other steps in block triangularization. At this time, for a general-purpose matching algorithm, we would advocate the use of the Duff and Wiberg algorithm because of its better asymptotic worst-case complexity and reasonable practical performance.

3.2 Coarse Decomposition

In the coarse decomposition, we use the maximum matching found in Phase 1 to partition the rows and columns. Initially, we include every column vertex in SC and then mark columns belonging to VC and HC. When the marking process terminates, all columns still marked SC will indeed belong to the set SC. A similar technique is used for the row vertices. The coarse decomposition algorithm is described below.

Algorithm 2. Coarse Decomposition

Step 0. [Initialize]
Include all column vertices in SC, all row vertices in SR.

Step 1. [Identify A_h]
U := {set of all unmatched column vertices};
for each $u \in U$ **do**
Include u in HC;
Perform a dfs from u,
including all rows reachable by alternating path in HR
and all columns reachable by alternating path in HC.
end for

```
Step 2. [Identify A_v.]
        W := {set of all unmatched row vertices};
        for each w ∈ W do
          Include w in VR;
          Perform a dfs from w,
          including all columns reached by alternating path in VC
          and all rows reached by alternating path in VR.
        end for
```

Consider the dfs from an unmatched column vertex. As noted before, since we are searching for alternating paths, the search from a row vertex is simple; we consider only the column matched to it. From a column vertex we search all rows adjacent to it. Hence we use the column-oriented adjacency lists, *adjrow* and *xadjr*, to find all rows adjacent to a given column and to find a column matched to a row from *rowset*. We use a pointer array into the column adjacency lists to mark how far we have progressed in the dfs from a column. Thus the adjacency list of each column in *HC* is searched at most once.

The first time a row vertex is reached by an alternating path from some unmatched column, we include the row in *HR*. Hence we can check if a row has already been visited by checking if it is in *SR* or *HR*.

By symmetry, by searching for rows and columns reached by alternating paths from unmatched row vertices, we can identify the sets *VR* and *VC*. The only difference is that now we need the row-oriented adjacency lists *adjcol* and *xadjc* and the array *colset* to identify rows matched to columns.

3.3 Fine Decomposition

In Phase 3, we further decompose the submatrices A_h, A_v, and A_s by finding the block diagonal forms of A_h and A_v and the block upper triangular form of A_s. The first task is accomplished by finding the connected components of the subgraphs G_h and G_v.

We find the connected components of each subgraph by a simple marking algorithm that uses dfs from vertices in the subgraph. We mark all vertices reachable by dfs from a start vertex; these belong to the same connected component. The marking procedure is repeated from an unmarked start vertex, as long as unmarked vertices exist. The dfs from each start vertex finds a connected component. Since we are no longer finding alternating paths, we must search the adjacency lists of both rows and columns. We need pointers to both column and row adjacency lists to ensure that each edge is examined at most twice: once in a column adjacency list and once in a row adjacency list. Hence this algorithm has $O(\tau)$ time complexity.

Row vertices in *HR* may be adjacent to columns in *SC* or *VC* also. Hence, when a column vertex is reached by dfs from a row in *HR*, we include it in this connected component of G_h only if the column belongs to *HC*. A similar test is used to ensure that only rows in *VR* are included in connected components of G_v.

The second task is the computation of the block upper triangular form of the square submatrix A_s. From Theorem 2.3, this can be accomplished by forming a directed graph G_d from G_s by directing the edges from column vertices to row vertices and shrinking matched edges to single vertices and by then finding the

strong components in G_d. It turns out that we can work with G_s directly, as will be described later.

Tarjan [34] has designed an algorithm using dfs to find the strong components of a directed graph in time linear in its edges. Duff and Reid [8] have implemented this algorithm, and their program is available as subroutine MC13D in the Harwell library. The program we have used is a variant of MC13D, modified to work with a sparse matrix represented by the SPARSPAK data structures. A good description of the program MC13D may be found in the book by Duff et al. [6], and we direct the reader there for details of the implementation.

One difference between our program and MC13D is that MC13D assumes that the matrix has already been permuted to have a zero-free diagonal. We do not permute the matrix A_s to make its diagonal zero-free, but work implicitly with G_d as follows. We search G_s using alternating paths beginning at row vertices and, from each row, taking a matched edge to the column vertex it is matched to. From a column vertex we search all edges leading to rows in SR.

We require a stack to put the rows we reach by alternating paths, a linked list to maintain the rows in the alternating paths, and an array *lowlink*, of size m, to identify the strong components. In addition, a pointer array is used to point to the next row to be examined in the dfs in a column's adjacency list, and another array represents the dfs numbers of the rows.

3.4 Output

We represent the btf of an $m \times n$ matrix A by means of three pointer arrays p, *cptr*, *rptr* and two integer arrays c and r. The pointer array p has length four, and the arrays *cptr* and *rptr* have length one greater than the total number of diagonal blocks in the btf of A. The integer arrays c and r have length n, m, respectively.

Columns in VC are listed in the array c in positions $cptr(p(1))$ to $cptr(p(2)) - 1$; similarly, columns in SC are listed in positions $cptr(p(2))$ to $cptr(p(3)) - 1$, and columns in HC in positions $cptr(p(3))$ to $cptr(p(4)) - 1$. A similar representation using p, *rptr*, and r is used for the rows.

The pointer arrays *cptr* and *rptr* are used to indicate the diagonal block structure of the submatrices A_v, A_s, and A_h. Columns in the first diagonal block of A_v are listed in array c in positions $cptr(p(1))$ to $cptr(p(1) + 1) - 1$, columns in the second block in $cptr(p(1) + 1)$ to $cptr(p(1) + 2) - 1$, and so on.

4. RESULTS

We implemented the algorithms described in the previous section in Fortran 77. We report computational results for several classes of rectangular matrices. Our code was executed on a Sun 4/260 running Sun Unix 4.0, and the f77 compiler was used to compile the code.

We report computational results on two collections of problems. When necessary, we have transposed the matrices to make them overdetermined. One set of problems was obtained from the constraint matrices of linear programming problems from the lp/data collection in the Netlib electronic software library. This set of 28 problems was selected arbitrarily from the lp/data collection; we

Table II. Linear Programming Test Problems

Problem	Rows	Cols.	Nonzeros	Density (%)
25FV47	1571	821	10400	0.8
FFFFF800	854	524	6227	1.4
BORE3D	315	234	1525	2.1
SCFXM1	457	330	2589	1.7
SCRS8	1169	490	3182	0.6
SHIP04L	2118	400	6332	0.8
SHIP04S	1458	400	4352	0.7
SHIP08S	2387	776	7114	0.4
SHIP12S	2763	1149	8178	0.3
SIERRA	2036	1227	7302	0.3
VTP.BASE	203	198	908	2.3
FORPLAN	421	161	4563	6.7
STANDGUB	1184	361	3140	0.7
STANDMPS	1075	467	3679	0.7
GANGES	1681	1309	6912	0.3
GFRD-PNC	1092	616	2377	0.4
PILOT4	1000	410	5141	1.3
SCAGR7	140	129	420	2.3
SCORPION	388	358	1426	1.0
AGG	488	163	2410	3.0
AGG2	516	302	4284	2.7
SEBA	1028	515	4352	0.8
RECIPE	180	91	663	4.0
SHELL	1775	536	3556	0.4
GROW7	301	140	2612	6.2
SCSD1	760	77	2388	4.1
SCTAP1	480	300	1692	1.2
SCTAP2	1880	1090	6714	0.3

report results for all of the problems tested. Pictures of the nonzero structure of some of these matrices are given by Lustig [29]. The problem parameters are shown in Table II. The problems in Table II are listed in an order that reflects the structure of their block triangular forms shown in Table III. The time needed to compute the various phases in the btf is shown in Table IV. Times for computing a maximum matching by Algorithm 1 and the Duff and Wiberg algorithm are shown.

An empty entry in the table for block triangular forms indicates that the submatrix corresponding to that column is missing in the btf. The number of diagonal blocks in each of the submatrices A_h, A_s, and A_v, and the row and column dimensions of these blocks are shown. Most of these linear programming constraint matrices have a nontrivial btf; the exceptions are the last five problems in Table III.

An empty entry in Table IV indicates that either the corresponding submatrix is absent from the btf or that the time required for that part of the computation was less than 0.01 seconds. Algorithm 1 finds a maximum matching, on the average, more than three times faster than the Duff and Wiberg algorithm. For these problems, the time to finely decompose the large submatrix A_v is often

Table III. Block Triangular Forms of lps.

	A_h			A_s		A_v		
Problem	Rows	Cols.	Blocks	Rows	Blocks	Rows	Cols.	Blocks
25FV47	3	6	3	45	43	1523	770	1
FFFFF800	52	63	1	112	112	690	349	1
BORE3D	8	12	3	50	44	257	171	1
SCFXM1	12	16	1	44	44	401	270	1
SCRS8	6	7	1	38	35	1125	445	1
SHIP04L	14	56	42	4	4	2100	340	4
SHIP04S	14	56	42	92	92	1352	252	4
SHIP08S	0	64	64	296	296	2091	416	1
SHIP12S	0	107	107	576	576	2187	466	1
SIERRA	80	90	5	100	25	1856	1037	1
VTP.BASE	95	122	2	42	42	66	34	1
FORPLAN	0	26	26	21	21	400	114	1
STANDGUB	48	64	8	125	77	1011	172	2
STANDMPS	48	64	8	124	76	903	279	1
GANGES				373	265	1308	936	1
GFRD-PNC				26	26	1066	590	1
PILOT4				8	8	992	402	1
SCAGR7				63	63	77	66	1
SCORPION				70	70	318	288	6
AGG				36	36	452	127	3
AGG2				60	60	456	242	3
SEBA						1028	515	25
RECIPE						180	91	12
SHELL						1775	536	1
GROW7						301	140	1
SCSD1						760	77	1
SCTAP1						480	300	1
SCTAP2						1880	1090	1

greater than the time to find the matching. This shows that the use of the various heuristics makes the matching algorithms fast in comparison to the connected components algorithm. Also, the time needed to compute the btf for these problems is quite small in comparison to the time that would be required by, say, a numeric factorization algorithm.

The second set of problems was obtained from the Boeing–Harwell test collection [7]. The rows, columns, and nonzeros in these problems are described in Table V. The four groups of problems indicate that they come from four different files in the collection.

The first group comes from geodetic surveys; the matrices in the second group are from least squares problems; and the last two groups are from economy models. Note that the problems in the last group are fairly dense. The block triangular forms of these problems and the total time to compute them are shown in Table VI. As before, an entry is empty if the corresponding submatrix is empty in the btf of the matrix. Not surprisingly, the survey matrices (except one) do

Table IV. Time (in hundredths of a second) Required to Compute the btf of lps on a Sun 4/260

	Time						
	Matching		Coarse decomp.		Fine decomp.		
Problem	Alg. 1	D-W Alg.	A_h	A_v	A_h	A_s	A_v
25FV47	3	20		5	1	1	11
FFFFF800	5	13		3	2	1	5
BORE3D	2	4		1		1	1
SCFXM1	3	6		1	1	1	2
SCRS8	3	11		2		1	5
SHIP04L	3	13	1	4	1	1	9
SHIP04S	2	9		3	1	1	6
SHIP08S	3	9	1	4	1	2	9
SHIP12S	3	11		5	2	4	10
SIERRA	5	20		5	1	1	10
VTP.BASE	2	2		1	1	1	
FORPLAN	2	6	1	1		1	4
STANDGUB	3	8		2	1	1	4
STANDMPS	3	8	1	2	1	1	4
GANGES	4	11		4		4	8
GFRD-PNC	2	8	1	2		1	4
PILOT4	2	8		3		1	6
SCAGR7	1	2				1	
SCORPION	2	3				1	2
AGG		2		1		1	3
AGG2	2	7		2		1	4
SEBA	3	8		3			5
RECIPE	1	2					1
SHELL	2	5		4			7
GROW7		2		1			3
SCSD1		4		2			4
SCTAP1	1	3		2			3
SCTAP2	3	8		4			11

not decompose at all. Finding the maximum matching accounts for about half the total time, and the fine decomposition of the larger submatrix (A_v for the first three groups, A_h for the last) requires almost all of the other half.

There are some common features in the block triangular forms of the matrices in these collections. Most of the matrices have one large submatrix, usually irreducible, that has most of the rows and columns of the matrix. For all of these problems, this large submatrix is A_v; even when this submatrix has several diagonal blocks, there is one large block, and the other blocks have at most two rows and at most two columns. The square submatrix A_s is almost always present, though it is small for all but a few problems. Several problems have empty rows or columns. Note that the last three matrices are square and structurally singular, and require the use of the more general Dulmage–Mendelsohn decomposition described here to compute their correct block triangular forms.

Table V. Test Problems from the Boeing–Harwell Collection

Problem	Rows	Cols.	Nonzeros	Density (%)
ASH219	219	85	438	2.4
ASH958	958	292	1916	0.7
ASH331	331	104	662	1.9
ASH608	608	188	1216	1.1
ABB313	313	176	1557	2.8
WELL1033	1033	320	4732	1.7
WELL1850	1850	712	8758	0.8
WM1	277	207	2909	5.1
WM2	260	207	2942	5.5
WM3	260	207	2948	5.5
BEAUSE	507	497	44551	17.7
BEAFLW	507	497	53403	21.2
BEACXC	506	497	50409	20.0
MBEAUSE	496	496	41063	16.7
MBEAFLW	496	496	49920	20.3
MBEACXC	496	496	49920	20.3

Table VI. Block Triangular Forms of Boeing–Harwell Problems and the Time (in seconds) Required to Compute Them

	A_h			A_s		A_v			
Problem	Rows	Cols.	Blocks	Rows	Blocks	Rows	Cols.	Blocks	Time
ASH219						219	85	1	0.03
ASH958						958	292	1	0.08
ASH331						331	104	1	0.03
ASH608						608	188	1	0.04
ABB313						313	176	2	0.05
WELL1033				16	14	1017	304	1	0.11
WELL1850				12	9	1838	700	1	0.22
WM1	2	27	1	49	49	228	158	27	0.06
WM2				35	35	225	172	1	0.06
WM3				27	27	233	180	1	0.06
BEAUSE	10	48	7	29	29	468	420	3	0.84
BEAFLW	8	45	7	11	11	488	441	8	0.90
BEACXC	0	48	48	8	8	498	441	18	0.87
MBEAUSE	2	51	49	26	26	468	419	7	0.75
MBEAFLW	0	48	48	8	8	488	440	12	0.87
MBEACXC	0	48	48	8	8	488	440	12	0.96

5. APPLICATIONS

In this section, we consider several applications of the Dulmage–Mendelsohn decomposition of a bipartite graph and the corresponding block triangular form of the associated matrix. Some applications only require the computation of the coarse decomposition; others can take advantage of the fine decomposition also.

Finding Node Separators from Edge Separators. In employing the divide and conquer paradigm, say in the context of designing parallel algorithms for sparse

matrices or graphs, it is necessary to find a small *vertex separator*, a set of vertices whose removal disconnects the graph into two or more components. One approach to computing a vertex separator is to first compute an *edge separator* (a set of edges whose removal disconnects the graph) and then to find a vertex separator from the endpoints of the edge separator.

Several algorithms for computing edge separators exist; for example, the Kernighan–Lin algorithm [24], spectral algorithms based either on the eigenvectors of the adjacency matrix (Barnes [1]), or on the eigenvectors of the Laplacian matrix of the graph [33].

Several strategies can be employed to find a vertex separator from an edge separator. The simplest is to choose the smaller set of endpoints of the edge separator; such a strategy has been employed by Gilbert and Zmijewski [19] in a parallel algorithm to partition sparse matrices for Cholesky factorization on a hypercube. Leiserson and Lewis [26] have implemented a heuristic algorithm to find small vertex separators from edge separators. We describe a strategy to compute the *smallest* vertex separator corresponding to a given edge separator. The idea is to choose a subset S of vertices from *both* endpoint sets such that every edge in the edge separator is incident on at least one vertex in S.

Let E_s be an edge separator that partitions a graph G into two disjoint vertex sets U and W. Let $B = (R, C, E_s)$ be the bipartite graph induced by E_s, where $R(C)$ is the set of endpoints of E_s in $U(W)$. A *minimum cover* S of B is a smallest set of vertices such that every edge in E_s has at least one endpoint in S. Since E_s is an edge separator of G, removing the set of vertices S from G will disconnect this graph, and thus S is the desired smallest vertex separator.

Maximum matchings and minimum covers in a bipartite graph are dual concepts, and a minimum cover S of B can be computed from its Dulmage–Mendelsohn decomposition. The set S can be chosen to be either $S = VC \cup SC \cup HR$ or $S = VC \cup SR \cup HR$. (It might be helpful to recall the definition of these sets in Figure 1.) This freedom in the choice of S can be used to partition the graph into nearly equal parts.

We computed vertex separators by this technique for several sparse matrices from edge separators obtained by a spectral algorithm described in [33]. The results are tabulated in Table VII. The time reported is in seconds on a Sun 3/75. For several problems, this strategy yields smaller vertex separators than the strategy of choosing the smaller endpoint set.

Improving Node Separators. A related problem is the improvement of node separators by matching techniques. Liu [27] has considered the following problem in the context of a constrained minimum degree ordering algorithm. Let S be a vertex separator that separates a graph $G\backslash S$ into two parts with vertex sets A, B. Suppose that $|A| > |B|$, and that there exists a subset $Y \subseteq S$ with $C = adj(Y) \cap A$ such that $|C| < |Y|$. Then $\bar{S} = S \cup C\backslash Y$ separates $G\backslash\bar{S}$ into two parts $\bar{A} = A\backslash C$ and $\bar{B} = B \cup Y$. Note that the new separator is smaller than the initial separator S, and that the smaller set, B, has increased in size at the expense of the bigger set, A.

The Dulmage–Mendelsohn decomposition can again be used to find a set Y with the required properties, if it exists. Let $D = adj(S) \cap A$, and consider the bipartite graph induced by the vertex sets S and D. If we identify D with the column set and S with the row set in Figures 1 and 2, then the required set

Table VII. Computing Vertex Separators from Edge Separators. (Times are in seconds on a Sun 3/75)

Problem	$\|R\|$	$\|C\|$	$\|E_s\|$	$\|S\|$	Time
cannes	229	184	712	91	0.04
dwt	30	29	87	28	0.06
pwr09	78	44	90	40	0.04
pwr10	279	181	365	160	0.08
stk13	392	322	4875	300	0.16

$Y = HC$ and $C = HR$. It can also be shown that this is the unique smallest set Y which maximizes the differences $|Y| - |adj(Y)|$ in the bipartite graph.

Liu [27] does not compute the Dulmage–Mendelsohn decomposition, but finds vertices in Y, one at a time, by an augmenting path search from vertices in S.

The Null Space Problem. We consider the problem of computing a sparse null basis of a sparse underdetermined matrix A [4, 18, 32]. An underdetermined A has the block lower triangular form,

$$A = \begin{pmatrix} A_v & & \\ X & A_s & \\ X & X & A_h \end{pmatrix},$$

where, as before, A_v is overdetermined, A_s is square, A_h is underdetermined, and 'X' denotes a possibly nonzero matrix of appropriate dimensions.

If A has full row rank, then all its rows are matched in a maximum matching, and its btf may have the submatrices A_h and A_s, but not A_v. Because A has full row rank, a null basis of A_h is a null basis of A, and hence, the submatrix A_s is of no interest in computing a sparse null basis. Algorithms in [4] compute a null vector by first identifying a dependent set in the matrix by means of matchings. It turns out that the submatrix formed by the columns and the nonzero rows of the dependent set has the Strong Hall Property.

If A does not have full row rank and if all its rows cannot be matched in a maximum matching of A, then the submatrix A_v may also be present in its btf. Algorithms to compute null bases by maximum matching methods have been designed in [4] and by Gilbert and Heath [18]. The btf of A enables these algorithms to partition the null basis computations as follows. Matching methods can be used to find dependent sets of columns and associated null vectors from A_h, but not from A_s or A_v; the remaining null vectors of A can be computed by numeric factorizations of each of the submatrices A_s and A_v.

The Linear Least Squares Problem. The problem of computing the orthogonal factors of a large, sparse, overdetermined matrix A arises in the sparse linear least squares problem. Algorithms for predicting the structure of the factors R and Q have been developed in recent years by George, Heath, Liu, and Ng [14, 16, 17]. The btf of A is of interest in this context.

If A is initially permuted to its btf, then only the diagonal blocks of the submatrices A_h, A_v, and A_s need be factored to compute the least squares solution, and this can lead to savings in computation and storage. Further, because each

submatrix that is factored has the Strong Hall Property, Coleman et al. [3] have shown that algorithms in current use for predicting the structure of the triangular factor R will not overestimate the storage required, since exact structural cancellation will not occur. George et al. [16] also assume that the matrix has been initially permuted to its btf and that their algorithms for predicting the structure of the orthogonal factor Q are applied to the diagonal blocks in the btf, since these have the Strong Hall Property.

Software Development. Several data structures used in mathematical software for sparse matrices simplify if we assume that the input matrix is irreducible or that the input graph is connected (strongly connected for directed graphs). For instance, tree data structures become forests if this assumption is violated. Our results in this paper show that sparse matrix test problems available from problem collections are often reducible. Computing the btf of a sparse matrix and providing the irreducible submatrices of A_h, A_v, and A_s as input to programs will help simplify development and testing and aid in the correctness of sparse matrix software.

ACKNOWLEDGMENTS

We wish to thank Tom Coleman and John Gilbert for their help in improving the presentation of the results in Section 2. The examples in Figures 1 and 3 are taken from John Gilbert's unpublished lecture notes. Thanks also to John Lewis for his comments on this paper.

REFERENCES

1. BARNES, E. R. An algorithm for partitioning the nodes of a graph. *SIAM J. Alg. Disc. Meth. 4* (1982), 541–550.
2. BUNCH, J. R., AND ROSE, D. J., EDS. *Sparse Matrix Computations.* Academic Press, New York, 1976.
3. COLEMAN, T. F., EDENBRANDT, A., AND GILBERT, J. R. Predicting fill for sparse orthogonal factorization. *J. ACM 33* (1986), 517–532.
4. COLEMAN, T. F., AND POTHEN, A. The null space problem II: Algorithms. *SIAM J. Alg. Disc. Meth. 8* (1987), 544–563.
5. DUFF, I. S. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Softw. 7* (1981), 315–330.
6. DUFF, I. S., ERISMAN, A. M., AND REID, J. K. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.
7. DUFF, I. S., GRIMES, R., AND LEWIS, J. Sparse matrix test problems. *ACM Trans. Math. Softw. 15* (1989), 1–14.
8. DUFF, I. S., AND REID, J. K. An implementation of Tarjan's algorithm for the block triangularization of a matrix. *ACM Trans. Math. Softw. 4* (1978) 137–147.
9. DUFF, I. S., AND WIBERG, T. Implementations of $O(n^{1/2}\tau)$ assignment algorithms. *ACM Trans. Math. Softw. 4* (1988), 267–287.
10. DULMAGE, A. L., AND MENDELSOHN, N. S. Coverings of bipartite graphs. *Can. J. Math. 10* (1958), 517–534.
11. DULMAGE, A. L., AND MENDELSOHN, N. S. A structure theory of bipartite graphs of finite exterior dimension. *Trans. Roy. Soc. Can. Sec. III 53* (1959), 1–13.
12. DULMAGE, A. L., AND MENDELSOHN, N. S. Two algorithms for bipartite graphs. *J. Soc. Ind. Appl. Math. 11* (1963), 183–194.
13. ERISMAN, A. M., GRIMES, R. G., LEWIS, J. G., POOLE, W. G., AND SIMON, H. D. An evaluation of orderings for unsymmetric sparse matrices. *SIAM J. Sci. Stat. Comput. 8* (1987), 600–624.

14. GEORGE, J. A., AND HEATH, M. T. The solution of sparse linear least squares problems using Givens rotations. *Lin. Alg. Appl. 34* (1980), 69–83.
15. GEORGE, J. A., AND LIU, J. W. H. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice Hall, Englewood Cliffs, N.J., 1981.
16. GEORGE, J. A., LIU, J. W. H., AND NG, E. G. Y. A data structure for sparse QR and LU factorizations. *SIAM J. Sci. Stat. Comput. 9* (1988), 100–121.
17. GEORGE, J. A., AND LIU, J. W. H. Compact structural representation of sparse Cholesky, QR and LU factors. In *Computing Methods in Applied Sciences and Engineering VII*, R. Glowinski and J.-L. Lions, Eds. Elsevier, North Holland, New York, 1986, 93–106.
18. GILBERT, J. R., AND HEATH, M. T. Computing a sparse basis for the null space. *SIAM J. Alg. Disc. Meth. 8* (1987), 446–459.
19. GILBERT, J. R., AND ZMIJEWSKI, E. A parallel graph partitioning algorithm for a message passing multiprocessor. *Int. J. Parallel Program. 16* (1987), 427–449.
20. GUSTAVSON, F. G. Finding the block lower triangular form of a sparse matrix. In *Sparse Matrix Computations*, J. R. Bunch and D. J. Rose, Eds. Academic Press, New York, 1976.
21. HOPCROFT, J. E., AND KARP, R. M. An $n^{2.5}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput. 2* (1973), 225–231.
22. HOWELL, T. D. Partitioning using *PAQ*. In *Sparse Matrix Computations*, J. R. Bunch and D. J. Rose, Eds. Academic Press, New York, 1976
23. JOHNSON, D. M., DULMAGE, A. L., AND MENDELSOHN, N. S. Connectivity and reducibility of graphs. *Can. J. Math. 14* (1962), 529–539.
24. KERNIGHAN, B. W., AND LIN, S. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J. 49* (1970), 291–307.
25. LAWLER, E. L. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart, and Winston, New York, 1976.
26. LEISERSON, C. E., AND LEWIS, J. G. Orderings for parallel sparse symmetric factorization. Talk at Third SIAM Conference on Parallel Processing for Scientific Computing (Tromso, Norway, 1987). SIAM, Philadelphia, 1987.
27. LIU, J. W. H. A graph partitioning algorithm by node separators. *ACM Trans. Math. Softw. 15* (1989), 198–218.
28. LOVASZ, L., AND PLUMMER, M. D. *Matching Theory.* North Holland, Amsterdam, 1986.
29. LUSTIG, I. J. An analysis of an available set of linear programs. Tech. Rep. SOL 87-11, Dept. of Operations Research, Stanford Univ., 1987.
30. MINC, H. *Nonnegative Matrices.* Wiley, New York, 1988.
31. PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, N.J., 1982.
32. POTHEN, A. Sparse null bases and marriage theorems. Ph.D. Thesis, Cornell Univ., Ithaca, N.Y., 1984.
33. POTHEN, A., SIMON, H. D., AND LIOU, K. P. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Math. Anal. Appl. 11* (1990), 430–452.
34. TARJAN, R. E. Depth-first search and linear graph algorithms. *SIAM J. Comput. 1* (1972), 146–160.

Received February 1989; revised June 1989; accepted November 1989.