SIAM J. ALG. DISC. METH.
Vol. 8, No. 4, October 1987

THE NULL SPACE PROBLEM II. ALGORITHMS*

THOMAS F. COLEMAN† AND ALEX POTHEN‡

Abstract. The null space problem is that of finding a sparsest basis for the null space (*null basis*) of an underdetermined matrix. This problem was shown to be NP-hard in Coleman and Pothen (this Journal, 7 (1986), pp. 527–537). In this paper we develop heuristic algorithms to find sparse null bases. A basis is computed by columns, i.e., by finding a null vector linearly independent of those previously obtained. The algorithms to compute null vectors have two phases. In the first combinatorial phase, a minimal dependent set of columns is identified by finding a matching in the bipartite graph of the matrix. In the second numerical phase, nonzero coefficients in the null vector are computed from this dependent set.

We have designed two algorithms: the first computes a fundamental basis (one with an embedded identity matrix), and the other, a triangular basis (one with an upper triangular matrix). We describe implementations of our algorithms and provide computational results on several large sparse constraint matrices from linear programs. Both algorithms find null bases which are quite sparse, have low running times, and require small intermediate storage. The triangular algorithm finds sparser bases at the expense of greater running times. We believe that this algorithm is an attractive candidate for large sparse null basis computations.

Key words. null basis, null space, sparse matrix, bipartite graph, matching, linear programming, nonlinear programming

AMS(MOS) subject classifications. 05, 15, 49, 65, 68

1. Introduction.

Currently successive quadratic programming is the most popular method to solve constrained nonlinear optimization problems. The quadratic programming subproblems are often solved by numerically stable null space algorithms. Thus designing efficient null space algorithms for large scale optimization problems is an area of intense research effort at present. One concern is that these algorithms require a sparse representation of the null space of the constraint matrix.

Let A be a $t \times n$ matrix of rank t. The Null Space Problem (NSP) (Pothen (1984), Coleman and Pothen (1986a)) is to find a basis N, with the fewest nonzeros, for the null space of A. For brevity, a basis for the null space will be called a *null basis*, and a column of a null basis will be called a *null vector*.

Two representations for the null basis N have been used so far in optimization algorithms. Wolfe (1962) proposed permuting the columns of A to obtain a $t \times t$ nonsingular matrix M such that $A = (MU)$, so that

$$N = \begin{pmatrix} B \\ I_{n-t} \end{pmatrix} \tag{1.1}$$

where $B \equiv -M^{-1}U$, and I_{n-t} is the $(n-t)$-dimensional identity matrix. We will call such a basis a *fundamental* null basis. An *explicit representation* of N is one in which the nonzeros in N are stored. In practice, N is represented *implicitly* by storing the LU factors of M, and a matrix-vector product such as Np is computed by solving a system of equations involving M. In the second representation, an LQ factorization of A is computed, and

* Received by the editors May 13, 1986; accepted for publication (in revised form) December 11, 1986. This research was supported in part by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under contract DE-AC02-83-ER10369. A preliminary version of this work was presented at the SIAM Conference on Applied Linear Algebra at Rayleigh, North Carolina, in April 1985.

† Department of Computer Science, Cornell University, Ithaca, New York 14853.

‡ Department of Computer Science, The Pennsylvania State University, University Park, Pennsylvania 16802.

the last $n - t$ columns of Q form an orthogonal null basis for A. This scheme is impractical for large scale problems since Q is likely to be quite dense.

A context in which an explicit representation of the null basis is required concerns the optimization of a nonlinear function subject to linear constraints. A null space method demands the matrix $N^T HN$, where H is the current Hessian or an approximation to it. If a subroutine to compute the gradient of the objective function is available, then it is possible to obtain a good approximation to HN by $n - t$ extra gradient evaluations (Gill, Murray and Wright (1981)), provided N is explicit. If N is explicit, and HN is sparse, then, in general, many fewer gradient evaluations will be needed if a sparse finite difference scheme is used (Coleman and Moré (1983), (1984)).

A second context arises from the recent work of Goldfarb and Mehrotra (1985) and Shanno and Marsten (1985) which extends Karmarkar's algorithm for linear programming. The crucial computational step in these works is the solution of large sparse linear least squares problems of the form

$$DNw = b,$$

where D is a diagonal matrix, and N is a null basis of the constraint matrix. Both groups suggest solving the linear systems using pre-conditioned conjugate gradients; however, a host of pre-conditioning strategies is lost if N is not explicit. For example, diagonal, incomplete Cholesky, and chordal (Coleman (1986)) pre-conditioners, all require an explicit null basis N. (Thapa (1984) discusses a variety of pre-conditioners available for optimization problems: most require explicit matrices.)

Hence, for the rest of this paper, we restrict ourselves to the study of sparse explicit representations of null bases.

Previous work by others. Recently much work has been done on computing sparse null bases. The "turnback" method for computing a null basis with a profile structure for equilibrium matrices in structural analysis was proposed by Topcu (1979). Kaneko, Lawo and Thierauf (1982) interpreted this algorithm from a matrix factorization point of view. Berry, Heath, Kaneko, Lawo, Plemmons and Ward (1985) refined this algorithm, implemented it using profile data structures, and tested it on several structural problems. Berry and Plemmons (1985) have implemented this algorithm on a HEP multiprocessor.

The turnback algorithm computes a QR factorization of A to identify a set of $(n - t)$ start columns. These are columns which are identified as linearly dependent in the factorization. Hence there is a null vector containing a start column and columns numbered lower than it in the matrix. Each null vector is computed by an algorithm which maintains a set of active columns, initially containing only a start column. Lower numbered columns are added to the active set, one by one, and a QR factorization of the active set is maintained. When the active set becomes dependent, the columns correspond to the nonzero components of a null vector. If the dependence involves the start column, the null vector is accepted. If not, the dependent column is rejected from the active set, and the process continued.

Null bases obtained by turnback are not fundamental; they have an embedded upper triangular matrix U_{n-t} of dimension $(n - t)$ with nonzero diagonal elements. Thus

$$N = \begin{pmatrix} B \\ U_{n-t} \end{pmatrix}, \tag{1.2}$$

and we call such bases *triangular* null bases.

Gilbert and Heath (1987) have implemented several algorithms to compute sparse null bases. One of these is the turnback algorithm using general sparse data structures

from Sparspak (George and Liu (1981)). Another is a matching based algorithm that computes triangular bases; we discuss this algorithm in § 4 of this paper.

Previous work by the authors. NSP was formulated in Coleman and Pothen (1986a). We briefly summarize definitions and major results from that paper that will be useful here.

A null vector of A can be obtained from a linearly dependent set of columns. We call such a set a *dependent set*. The coefficients of the linear combination correspond to the values of the nonzero components of the null vector. A minimal dependent set of columns of A is a *circuit*. We proved that only null vectors that correspond to circuits could be columns in a sparsest null basis.

Sparsest null bases were characterized by a greedy algorithm that augmented a partial basis by a sparsest null vector independent of those previously chosen. Despite this result, finding a sparsest null basis is computationally an intractable problem since it is NP-hard. Computing a sparsest fundamental null basis is also NP-hard.

We addressed the question if sparsest null bases could be characterized to have some particular zero-nonzero structure (*structure*). It is known that a sparsest null basis may not be fundamental. We showed that a set of k vectors is linearly independent for all possible numeric values of its nonzeros if and only if it has an embedded upper triangular matrix of dimension k. Yet we do not know if we can always restrict a sparsest basis to be triangular. Nevertheless, restriction of the structures to fundamental and triangular bases makes it easy to ensure linear independence of the null vectors.

The relation between a triangular and a fundamental basis is an interesting one. Since a triangular null basis has the structure in (1.2), if A is partitioned to conform to N as $A = (MS)$, then we have $B = -M^{-1}SU_{n-t}$. Hence

$$N = \begin{pmatrix} -M^{-1}S \\ I_{n-t} \end{pmatrix} U_{n-t}.$$

Thus a triangular basis is obtained from a fundamental basis by postmultiplying with an upper triangular matrix. From matrix algebra alone, it is hard to see that triangular bases can be sparser than fundamental bases. The results in this paper show that judiciously constructed, they are sparser.

Outline of this paper. In this paper we report on the design and implementation of algorithms to compute fundamental and triangular null bases. A null basis is computed by repeatedly executing an algorithm to compute a null vector. Since sparsest null bases are characterized by the greedy algorithm, a heuristic strategy of computing the basis by repeatedly finding sparse null vectors is justified.

The algorithm to compute a null vector has two phases: in the first combinatorial phase, we identify the nonzero positions in the null vector. The nonzeros in each null vector corresponds to columns of A in a circuit. In a second numeric phase, numeric values of the nonzeros are computed.

In § 2 we design a circuit algorithm that finds a dependent set from a maximum matching of A. If the matrix satisfies a nondegeneracy assumption called the weak Haar property, then this dependent set is a circuit. We also show that any circuit of A can be found from an appropriate matching. The matching theory needed to understand this paper is introduced as needed in this section.

We use the circuit algorithm to find a fundamental null basis in § 3. All circuits in a fundamental basis are computed from one fixed matching in the matrix. We report on an implementation and on our computational results.

Section 4 describes the triangular algorithm to compute triangular null bases. Here a modified circuit algorithm which chooses columns to add to a start column such that a sparse circuit is obtained is used to find circuits. The algorithm is guided in its choice of columns by a matching which is constructed simultaneously. Thus each circuit is obtained from a separate matching.

Ensuring the correctness of the triangular algorithm is a subtle issue that involves some matching theory; we prove that if a complete matching is maintained in the nonstart columns in the matrix, correctness can be assured. This "outer" matching is distinct from the matching from which a circuit is obtained.

In § 5 an implementation of the triangular algorithm is described and our results are discussed. We compare our results with the results of Gilbert and Heath (1987).

In § 6 we list some results on sparse orthogonal null bases we have obtained in Coleman and Pothen (1986b), summarize our work, and make some additional remarks.

By convention a term is in *slanted* font when it is being defined. We also denote the set operations $A \cup \{b\}$, $A \backslash \{b\}$, and $A \cup \{b\} \backslash \{c\}$ by $A + b$, $A - b$, and $A + b - c$, respectively.

2. A circuit algorithm. A sparse null vector is computed in two phases. In the first phase, a circuit is identified from a matching in the matrix. In the second phase, the nonzero coefficients of a null vector are computed by solving a system of equations. We proceed to introduce the matching theory needed to design a circuit algorithm.

The *bipartite graph* $G(A)$ of the matrix A has a row vertex corresponding to each row of A, and a column vertex corresponding to each column of A. An edge joins a row vertex to a column vertex if and only if the corresponding matrix element is nonzero. The structure of a matrix and its bipartite graph are shown in Fig. 2.1. The symbols "×", and "⊗" denote nonzeros; the rest are zeros.

A *matching* in A is a set of nonzeros of A such that no two elements in the set are chosen from the same column or the same row. A matching of A corresponds in $G(A)$ to a set of edges no two of which are incident on a common vertex. A matching in the matrix of Fig. 2.1 is shown by circled nonzero elements; in the bipartite graph the edges in the matching are drawn with thick lines. A vertex is *matched* if it is an endpoint of an edge in a matching. A vertex that is not matched is *unmatched*. The matching $\mathcal{M}$ in the figure has maximum cardinality, and hence is a *maximum matching* of A. The *match-*

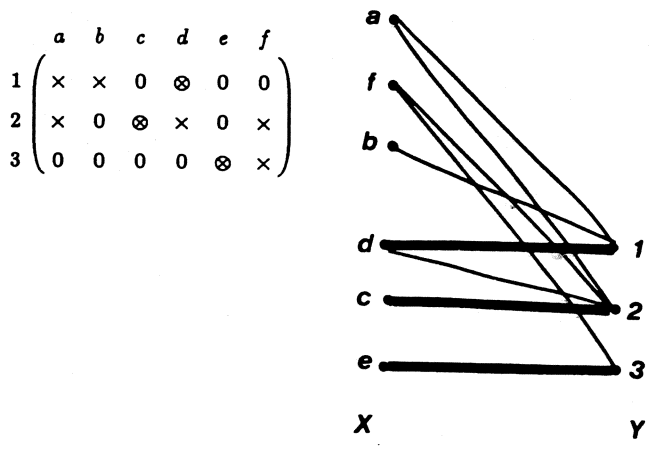

FIG. 2.1. *The structure of a matrix, its bipartite graph, and a matching.*

ing number, $m(A)$, is the cardinality of a maximum matching of A. A matching in which all the rows are matched is a *complete* (row-perfect) matching in A. A matching in which all rows and columns are matched is a *perfect* matching.

There exist several polynomial time algorithms to find maximum matchings in bipartite graphs. Let τ denote the number of nonzeros in A. The theoretically fastest known algorithm is due to Hopcroft and Karp (1973), and has time complexity $O(t^{1/2}\tau)$. Duff (1981) prefers an $O(t\tau)$ algorithm which he finds is faster in practice. Good discussions of matching algorithms may be found in Papadimitriou and Steiglitz (1982), and Lawler (1976).

The following two propositions are well known; Bondy and Murty (1976) have a proof of the first. The matrix A has the *Hall Property* (HP) if every subset of its rows has nonzeros in at least as many columns.

PROPOSITION 2.1 (Philip Hall). *A has a complete matching if and only if it has the Hall property.*

PROPOSITION 2.2. *The matching number of a matrix is greater than or equal to its rank.*

From Proposition 2.2, a matrix with rank t has a complete matching. A stronger condition on A is the Strong Hall Property (SHP). The matrix A has the *Strong Hall property* if every subset of $0 < k < n$ rows has nonzeros in at least $k + 1$ columns. (Thus when $t < n$, every set of $k \leqq t$ rows has nonzeros in $k + 1$ columns, and when $t = n$, every set of $k < n$ rows has nonzeros in $k + 1$ columns.) SHP is the same property as irreducibility. The terms HP and SHP are due to Coleman, Edenbrandt and Gilbert (1986).

A complete matching $\mathcal{M}$ of A partitions the columns of A into two sets: M, the set of matched columns, and U, the set of unmatched columns. In Fig. 2.1, $M = \{c, d, e\}$ and $U = \{a, f, b\}$. We now show that for a column $u \in U$, we can construct a circuit of A containing u by an "alternating path algorithm."

A *path* in a graph is a sequence of distinct vertices $v_1, \cdots, v_k$, where (v_{i-1}, v_i) is an edge of the graph, for $1 < i \leqq k$. An *$\mathcal{M}$-alternating path* is a path whose edges are alternately chosen from the matching $\mathcal{M}$ and outside $\mathcal{M}$. In Fig. 2.1 the sequence of edges $(b, 1)$, $(1, d)$, $(d, 2)$, $(2, c)$ is an $\mathcal{M}$-alternating path in A. We say that c and d are reachable from b by $\mathcal{M}$-alternating paths, and indicate this by $b \overset{M}{\rightarrow} c$ and $b \overset{M}{\rightarrow} d$.

An *augmenting path* is an alternating path which begins and ends with unmatched vertices. By making matched edges along an augmenting path unmatched, and vice versa, the size of the matching can be increased by one.

For $u \in U$, the following algorithm constructs a dependent set $n(u)$ containing u; this is a circuit if A has the Weak Haar Property (WHP). A matrix has the *weak Haar property* if every set of columns C satisfies rank $(C) = m(C)$. This assumption ensures that $n(u)$ will be a circuit for all "general" numeric values of the columns of A. For a particular set of numeric values of the nonzeros of A, numerical cancellations may occur, in which case the set $n(u)$ will contain a circuit. (The definitions of HP, SHP and WHP are tabulated in Table 1 for easy reference.)

TABLE 1
Summary of properties.

Hall Property (HP)	every subset of k rows has nonzeros in at least k columns
Strong Hall Property (SHP)	every subset of k rows has nonzeros in at least $k + 1$ columns
Weak Haar Property (WHP)	every subset of columns C has $m(C)$ = rank (C)

THE CIRCUIT ALGORITHM. Given a matrix A with WHP, a complete matching $\mathcal{M}$, and an unmatched column $u \in U$, this algorithm finds a circuit $n(u)$.

Follow all $\mathcal{M}$-alternating paths from u, adding columns visited to $n(u)$.
Thus $n(u) = u + \{v \in M : u \xrightarrow{M} v\}$.

From Fig. 2.1 it is easy to see that $n(a) = \{a, d, c\}$, $n(b) = \{b, d, c\}$, and $n(f) = \{f, c, e\}$. The set $n(u)$ can be constructed in $O(\tau)$ time by a depth first search.

THEOREM 2.3. *The set of columns $n(u)$ is a circuit if A has WHP.*

Proof. Let C be the set of columns in the dependent set $n(u)$, and let C have nonzeros only in the row set R. For ease of notation, denote by B the submatrix A_{RC}.

We first show that B has SHP. Consider any subset S of k rows of B. S is matched in $\mathcal{M}$ to k columns, all of which are in C. If the unmatched column u has a nonzero in any of the rows in S, then S has nonzeros in at least $k + 1$ columns.

Suppose that u has no nonzero in S. Since rows in S are reachable from u by $\mathcal{M}$-alternating paths, there must exist a column, matched to a row outside S, with a nonzero in S. Again, S has nonzeros in at least $k + 1$ columns.

Let b be any column in B. Since B has SHP, $B - b$ has HP. By Proposition 2.1, $B - b$ has a complete matching of size $|R|$. Since A has WHP, the rank of $B - b$ is $|R|$, and so the columns in $B - b$ are independent. Since B is dependent, it follows that $n(u)$ is a circuit. □

Let $\tilde{C}$ denote the submatrix of columns in $C - u$ and rows in R, and let $\tilde{u}$ denote the components of u corresponding to rows in R. The coefficients of the null vector can be computed by solving

$$\tilde{C}x = -\tilde{u},$$

and then choosing

$$n(u)_i = \begin{cases} x_i & \text{if } i \text{ corresponds to a column in } \tilde{C}, \\ 1 & \text{if } i \text{ corresponds to } u, \\ 0 & \text{otherwise.} \end{cases}$$

Suppose that $\tilde{C}$ does not have WHP. Since it has a perfect matching by construction, it is rank deficient. Hence it may not be possible to express $\tilde{u}$ as a linear combination of columns in $\tilde{C}$. In this case, the coefficient of u in the null vector is zero, and we say that *the dependence in $n(u)$ does not involve u.* However, it is possible to choose a column m which has a nonzero coefficient in the null vector. Thus a null vector $n(m)$ with a nonzero component corresponding to m is obtained.

COROLLARY 2.4. *If $n(u)$ does not have WHP, it contains a circuit which can be identified by a numeric factorization.* □

We now prove that the converse of Theorem 2.3 is true.

THEOREM 2.5. *Every circuit of a matrix A with WHP can be constructed by the circuit algorithm from some maximum matching $\mathcal{M}$ of A.*

Proof. Let C be the set of columns in a circuit, and let R and B be as in the proof of Theorem 2.3. Denote $|C|$ by c, and distinguish any one column of B as u. We claim $B - u$ has HP.

Suppose not. Then R has a subset of k rows adjacent to fewer than k columns, for some k. The columns and rows of the submatrix B can be permuted to the structure in Fig. 2.2. The submatrix B then contains a dependent set of columns of size $c - k$, violating the minimality of C. Hence $B - u$ has HP, and by Proposition 2.1, it has a complete matching $\mathcal{M}_1$. Partition A as shown in Fig. 2.3.

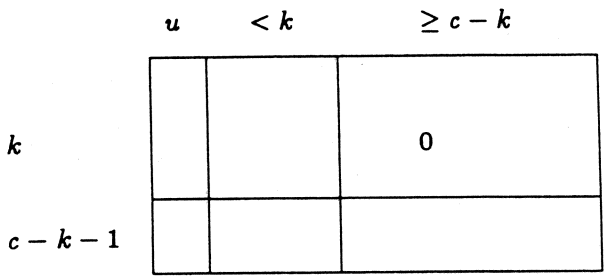

FIG. 2.2. *The submatrix B.*

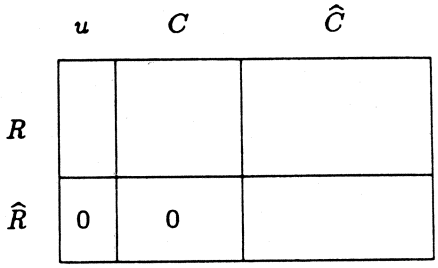

FIG. 2.3. *A partition of A.*

In any matching of A, the row set $\hat{R}$ can match only to the column set $\hat{C}$. Let $\mathcal{M}_2$ be a maximum matching of the submatrix $A_{\hat{R}\hat{C}}$. The required maximum matching is $\mathcal{M}_1 \cup \mathcal{M}_2$. □

3. Fundamental null bases. We now develop an algorithm to compute fundamental null bases using the circuit algorithm.

THE FUNDAMENTAL ALGORITHM.

1. [initialize] Let N be the empty set;
2. [match]
   Find a complete matching $\mathcal{M}$ of A;
   partition the columns: $A = (MU)$;
3. [construct basis]
   for each $u \in U \rightarrow$
   construct $n(u)$ by the circuit algorithm;
   solve for the coefficients in $n(u)$;
   Augment the null basis N with the computed null vector;
   rof

When A has WHP, by Theorem 2.3, each set $n(u)$ is a circuit. Further, since an unmatched column u is contained only in the circuit $n(u)$ by construction, N is a fundamental null basis. Thus the algorithm is correct in this case.

Step 3 of the fundamental algorithm can be modified to reduce its complexity. With the partition $A = (MU)$, the fundamental basis has the structure in equation (1.1), where $B \equiv -M^{-1}U$. Thus when M has full rank, the coefficients of each null vector $n(u)$ can be obtained by solving a system of the form $Mx = -u$. Hence we do not need to identify columns in $n(u)$ by following alternating paths. Also, the $(n - t)$ matrix factorizations can be replaced by one.

This observation also shows that when A does not have WHP, the algorithm can fail to compute all the $(n - t)$ linearly independent null vectors in a basis. Corresponding to each fundamental basis, there is an associated partition of the columns of A into M and U. Since B satisfies the equation $MB = -U$, when M does not have full rank, it may not be possible to express a column u as a linear combination of the columns in M.

Thus when A does not have WHP, a fundamental basis can be computed only when M has full rank. Hence we choose M by a matching, but ensure that M has full rank when we factor it to compute the null vectors. If it is rank-deficient, we reject the dependent columns in M from the matching, and find a new maximum matching. This strategy will ensure correctness; and will always succeed when A has full row rank.

For some of the problems reported here, the submatrix M chosen by a matching was indeed rank deficient. The number of dependent columns was almost always equal to one or two out of a few hundred columns; the largest we observed was five.

Details of implementation. Since computing a sparsest fundamental null basis is NP-hard, heuristic strategies have to be employed to find sparse bases. Our strategy is to assign costs to the columns of A, and to choose a column of minimum cost to match to a row.

The cost of a column c is the number of nonzeros in it. To justify this, observe that for a matrix with WHP, the number of nonzero rows in a circuit is one smaller than its number of columns. Thus a sparse circuit has few nonzero rows. Hence the cost of a column c indicates that any circuit containing c must have at least this number of additional matched columns in it. The cost of a matching is the sum of the costs of the matched columns. Ties were broken in favor of lower numbered columns.

Our weighted maximum matching routine is derived from MC21A, Duff's algorithm for finding a maximum matching in a matrix (Duff (1977)). A maximum matching is obtained by matching the rows one by one. At one step of the matching algorithm, we search for a column to match to an unmatched row. From the given row, a depth first search is performed through alternating paths to visit every unmatched column that could be reached by such a path. The cheapest of these columns is chosen.

The solution of the linear systems to compute the coefficients of the null vector is accomplished by using the LUSOL package of Gill, Murray, Saunders and Wright (1986). This package is presently a part of MINOS (Murtagh and Saunders (1983)). The LUSOL routines draw on the work of Reid (1976), (1982) on sparse LU factorizations of unsymmetric matrices. Gaussian elimination with row and column pivoting is performed such that $M = LU$, where the matrix PLP^t is lower triangular, PUQ is upper triangular, and P, Q are permutation matrices. Markowitz's criterion is used to select the pivot element, subject to a bound on the size of elements in L for numerical stability. Two triangular systems are solved to compute the null vector from the factors; we call this a *solve*. Parameters in LUSOL were set at their default values.

The matching algorithm has complexity $O(t\tau)$; Duff (1981) reports an $O(t) + O(\tau)$ experimental behavior. Since the dimension of M is t, the cost of factoring it in step 3 could be $O(t^3)$, and the $(n - t)$ solves could cost $O((n - t)t^2)$ operations. However, since M is sparse, a more realistic cost should be about $O(t^2)$ for the factorization and $O((n - t)t)$ for the solves.

Storage requirements of the algorithm are dominated by the storage required for A and the null basis N. The matrix A is stored in Sparspak column oriented data structures (George and Liu (1981)). Nonzeros and row indices are stored in column major order. For the use of matching routines, column indices of the nonzeros are stored in row major

order. These require a double precision array of length τ, two integer arrays of length τ, and integer arrays of length $n + 1$ and $t + 1$. Let $\tau(M)$ denote the number of nonzeros in M; this is smaller than τ. LUSOL needs the submatrix M stored as an element list with parallel integer arrays for row and column indices. The factorization is stored in the data structure for M, for which a minimum length of $\tau(M) + 4t$ is recommended. The null basis N is stored in Sparspak column oriented data structures of length equal to the number of nonzeros in the basis.

Results. We implemented the fundamental algorithm in FORTRAN 77; our experimental code, BASIS, is structured and modular, and we believe it represents a careful and efficient implementation. The program was run on a VAX 11/780 (with floating point coprocessor) under Berkeley 4.2 Unix at Penn State's Computer Science Department. The f77 compiler was used to compile the code.

Constraint matrices from linear programming problems were used for tests, and are shown in Table 2. The first, *murty*, was taken from Murty (1983), and all the others were supplied to us by Dr. Michael Saunders. The nonzero matrix elements were stored in double precision. Our results are tabulated in Table 3. The algorithm found bases comparable in sparsity to the input matrices for all problems except *brandy*, for which there was a four fold increase in density. This seems to be caused by the restriction to fundamental bases, as will be seen in the next section.

The total time (seconds) reported includes the time needed to find a maximum matching, compute the LU factors of M, and solve for the null vectors. The matchings were found quite fast, and the relatively larger times for the problems *brandy*, *capri* and *etamacro* were caused by dependence in M which necessitated finding a second matching. This step can be speeded up if the current matching is updated instead of finding a new matching as we have done.

Both *murty* and *israel* had embedded identity matrices of dimension t; thus these null bases were anomalously easy to find. The times reported for these problems should therefore be considered low.

The time for the factorization phase was surprisingly low. This is due to the high sparsity in M as a result of the column selection strategy in the matching algorithm, and the efficient method for computing sparse factors via LUSOL. Solving for the coefficients accounted for most of the time needed by the algorithm; solving for each null vector took less than a tenth of a second, but the large number of null vectors caused the large

TABLE 2
Test problems.

Problem	Rows	Cols	Nonzeros	Density (%)
murty	12	30	56	15.6
afiro	27	51	102	7.4
adlittle	56	138	424	5.5
share2b	96	162	777	5.0
share1b	117	253	1179	4.0
beaconfd	173	295	3408	6.7
israel	174	316	2443	4.4
brandy	193	303	2202	3.8
e226	223	472	2768	2.6
capri	271	482	1896	1.5
bandm	305	472	2494	1.7
stair	356	614	4013	1.8
etamacro	400	816	2537	0.8

TABLE 3
Fundamental null bases.

	Null basis				Time (seconds)			
Problem	Rows	Cols	Nonzeros	Density (%)	Total	Match	Factor	Solve
murty	30	18	62	11.5	0.6	.02	.07	0.5
afiro	51	24	112	9.2	0.9	.05	.08	0.8
adlittle	138	82	500	4.4	3.5	.07	0.2	3.2
share2b	162	66	736	6.9	4.6	0.1	1.0	4.0
share1b	253	136	2264	6.6	13.7	0.7	1.7	11.3
beaconfd	295	122	1789	5.0	13.9	0.7	1.1	11.9
israel	316	142	2411	5.4	12.1	.02	0.4	11.6
brandy	303	110	4758	14.3	22.8	1.8	2.3	18.6
e226	472	249	3449	2.9	20.3	0.3	0.6	19.2
capri	482	211	3478	3.4	27.6	4.5	2.5	20.3
bandm	472	167	2306	2.9	19.5	0.7	0.1	17.4
stair	614	258	5378	3.4	35.7	1.2	1.2	33.2
etamacro	816	416	3929	1.2	39.1	2.8	2.5	33.8

time requirement. We conclude also that the numerical phase of the algorithm dominates the combinatorial phase in computational time required.

The numerical quality of each null vector was checked by computing the residual of the system of equations used to find a null vector. In all cases, this was below machine precision. Condition numbers were estimated for constraint matrices and null bases of the first eight problems in Table 2 by the LINPACK condition estimator. The estimates for the null bases were lower than the estimates for the constraint matrices, except for share1b; here the null basis had a condition estimate of approximately 10^6, about ten times that of the constraint matrix.

4. The triangular algorithm. We now describe the *triangular algorithm* that computes a triangular null basis by matching. The diagonal elements of the triangular basis correspond to a set of $n - t$ *start columns* in A. The algorithm computes a circuit containing each start column. Linear independence of the set of circuits follows from the structure of N.

Throughout this section we do not assume that A has WHP; hence the set $n(u)$ found from a matching will not necessarily be a circuit, but only a dependent set. The corresponding null vector may not have a nonzero coefficient corresponding to u. By a *null vector* $n(u)$ we mean a null vector with a nonzero component corresponding to u. In the description of the triangular algorithm, the remedial actions necessary in the absence of WHP are included.

THE TRIANGULAR ALGORITHM. Given a complete matching $\mathcal{M}$ in a matrix A, and a partition into matched columns M and unmatched columns U, this algorithm computes a triangular null basis N. The set S is the set of columns which have already been used as start columns.

$S := \varnothing$;
while $U \neq S$ **do**
    Choose a column $u \in U - S$;
    Construct the dependent set $n(u)$ from columns in $A - S$;
    Solve for the corresponding null vector;

```
if dependence involves u
        then { the null vector is n(u)}
             S := S + u; N := N + n(u);
        else {u has a zero coefficient in the null vector}
             find a column m involved in the dependence;
             Let n(m) be the associated null vector;
             S := S + m; N := N + n(m);
             if m ∈ M then {update the matching M}
                                let r be the row matched to m;
                                M := M − (r, m);
                                Augment M by matching r
                                to a column in A − S;
                                Let v be the newly matched
                                column;
                                M := M − m + v; U := U +
                                m − v;
                           fi
        fi
od
```

Description of the algorithm. Let $S = \{s_1, \cdots, s_{i-1}\}$ be a set of start columns for which null vectors $\{n(s_1), \cdots, n(s_{i-1})\}$ have been computed. Columns in S will not be used in any of the null vectors to be computed in the future. The triangular algorithm maintains the invariant that $A - S$ has a complete matching $\mathcal{M}$. (This matching may change in the course of the algorithm.) The matching $\mathcal{M}$ partitions the columns of A into a set M of matched columns and a set U of unmatched columns, and $S \subseteq U$.

There is a great deal of freedom in how a dependent set $n(u)$ is constructed in this algorithm. As in the fundamental algorithm, we could employ the circuit algorithm to find $n(u)$ from the matching $\mathcal{M}$. Or, we could find $n(u)$ from *another* matching in $A - S$, with a view toward obtaining a sparse null vector. Indeed, we choose to do the latter.

Later in this section, we present a modified circuit algorithm that, given u, forms $n(u)$ by *choosing* columns in $A - S$ by simultaneously constructing a matching in $n(u)$. This matching is different from the complete matching $\mathcal{M}$. Thus for each start column u, a different matching in $A - S$ is constructed. This permits more intelligent choices in the column selection strategy to achieve sparsity. It is still essential to maintain a complete matching $\mathcal{M}$ in $A - S$ to ensure the correctness of the triangular algorithm.

Initially the complete matching $\mathcal{M}$ partitions the columns into M and U, and S is empty. For a column $u \in U - S$, a dependent set $n(u)$ is constructed by a matching algorithm from the columns in $A - S$. The corresponding null vector is computed as described in § 2, by a numeric factorization. If u is involved in the dependence, then N is augmented with the vector $n(u)$, and u is added to the set S. Otherwise, we can identify a column m with a nonzero coefficient in the null vector, and we obtain a null vector $n(m)$. The column m is added to the set S, and N is augmented with the vector $n(m)$.

If $m \in M$, then the row r matched to m in $\mathcal{M}$ has to be matched to another column in $A - S$ to maintain the invariant. This is accomplished by augmenting $\mathcal{M} - (r, m)$ to a complete matching, and updating the sets M and U.

It is easily seen that a triangular basis is obtained if the null vectors are arranged in the reverse order in which they are computed.

The modified circuit algorithm. We describe the algorithm used to find the dependent set $n(u)$. The modified circuit algorithm is a variant of an algorithm proposed by Gilbert

and Heath (1986), based on the matching theory developed in this paper. First, we describe our version.

THE MODIFIED CIRCUIT ALGORITHM. Given a column u, this algorithm finds a dependent set $n(u)$. Here S is the set of start columns for which null vectors have been computed by the triangular algorithm, C is the set of active columns, and R is the set of active rows.

$C := \{u\}$;
$R :=$ {rows in which u has nonzeros};
while there is an unmatched row $r \in R$ **do**
    Find an augmenting path from an unmatched active row r to an inactive column $c \in A - S$;
    Augment by adding r and c to the matching;
    $C := C + \{c\}$;
    $R := R +$ {inactive rows in which c has nonzeros};
    od

This algorithm identifies an *active submatrix* formed from a set R of *active rows* and a set C of *active columns* such that A_{RC} is dependent. Initially, the only active column is the column u, and the active rows are the rows in which u has nonzeros. A queue of active rows is maintained by the algorithm. At each step, a column is chosen to match to a row in the queue. The column is added to the set of active columns, and inactive rows in which the column has nonzeros are made active and added to the queue. At termination the active rows are perfectly matched to the active columns (excluding u). If $n(u)$ has WHP, by Theorem 2.3, the active columns form a circuit.

Assume that in case of failure to find a column c to match to a row, the modified circuit algorithm terminates. We can prove that this will not happen; i.e., the modified circuit algorithm will not terminate without finding a dependent set $n(u)$.

THEOREM 4.1. *Let $A - S$ have a complete matching $\mathcal{M}$ which partitions A into M and U, with $S \subset U$. Then for a column $u \in U - S$, the modified circuit algorithm will find a dependent set $n(u)$ containing columns from $A - S$.*

Proof. If the algorithm succeeds in finding a column to match, the size of C increases by one in each iteration of the **while** loop. If it fails, the algorithm terminates. In either case, termination is assured.

If all rows in R are matched at termination, then they are matched to columns in $C - u$, and from Theorem 2.3, the columns in C form a dependent set. Hence assume that the algorithm terminates with a matching $\mathcal{M}_1$ which matches columns in $C - u$ to a subset of rows in R. Let $R_u \subseteq R$ denote the set of unmatched rows which cannot be matched by finding augmenting paths.

Let $\mathcal{M}_2$ denote the edges in the complete matching $\mathcal{M}$ incident on the rows in R. By Theorem 4.1 of Lawler (1976, Chap. 5) (also Gale and Hoffman (1982)), it is possible to find a matching $\mathcal{M}_3$ from $\mathcal{M}_1$ and $\mathcal{M}_2$ in which all rows in R and all columns in $C - u$ are matched. Thus it is possible to augment the matching $\mathcal{M}_1$ by matching rows in R_u, and this is a contradiction. □

Correctness of the triangular algorithm. We establish the correctness of triangular algorithm next. In view of Theorem 4.1, we need prove only that it is possible to maintain the invariant of the algorithm, after a start column is chosen and a null vector is computed.

THEOREM 4.2. *Let $1 \leqq i \leqq n - t$, and $S = \{s_1, \cdots, s_{i-1}\}$ be a set of start columns for which null vectors $n(s_1), \cdots, n(s_{i-1})$ have been computed. Let $A - S$ have a complete*

matching $\mathcal{M}$. There exists a column $s_i \in A - S$ from which the triangular algorithm computes a null vector $n(s_i)$ from the columns in $A - S$, such that $A - (S \cup s_i)$ has a complete matching.

Proof. The modified circuit algorithm finds a dependent set $n(u)$ from any column u which is unmatched in $\mathcal{M}$. If the dependence involves u, then $s_i := u$, and the result is true. Otherwise, let m be a column in $n(u)$ involved in the dependence. Now $s_i := m$, and $n(m)$ is the null vector obtained.

Let M denote the columns matched in $\mathcal{M}$, and U the unmatched columns. Denote $S \cup s_i$ by $\hat{S}$. To maintain the invariant, there are two cases to consider.

Case 1. $m \in U - S$. Then M remains a completely matched set of columns in $A - \hat{S}$.

Case 2. $m \in M$. In the matching found by the modified circuit algorithm, the set of columns $n(u) - u$ is perfectly matched to the set of rows in which $n(u)$ has nonzeros. Call this set of rows R. (See Fig. 4.1.) Then from the proof of Theorem 2.5, the set of columns $n(u) - m$ can be perfectly matched to R. Since $n(u) - m \subseteq A - \hat{S}$, it is possible to match all rows in R to columns in the latter set.

Let $\hat{R}$ denote the rest of the rows of A. Then columns in $n(u)$ have zeros in the rows in $\hat{R}$, and hence a row in $\hat{R}$ cannot, in any matching, match to any of these columns. Thus columns matched in $\mathcal{M}$ to rows in $\hat{R}$ are disjoint from columns in $n(u) - m$. It follows that all rows in this set can be matched to columns in $A - \hat{S}$. Hence $A - \hat{S}$ has a complete matching.

Let r be the row matched in $\mathcal{M}$ to m. It follows from the correctness of the maximum matching algorithm that $\mathcal{M} - (r, m)$ can be augmented to a complete matching by matching r to a column in $A - \hat{S}$. □

Gilbert and Heath use a version of the modified circuit algorithm as a component in their matching algorithm (GHM) to find triangular bases. The major difference is that they maintain a QR factorization of the active submatrix as each column is being added. This helps them terminate the algorithm when a numerical dependence is detected, even when all active rows have not been matched.

We find a dependent set by matching methods alone, and then solve for the coefficients in the null vector by a sparse LU factorization of the submatrix A_{RC}. There are two advantages to such a choice. Since all the columns and rows in the dependent set are known, the row and column ordering strategies of a sparse matrix factorization can be used to keep the LU factors sparse. More important, a sparse matrix storage scheme can be used to store the nonzeros in the factors, thus keeping intermediate storage needed low.

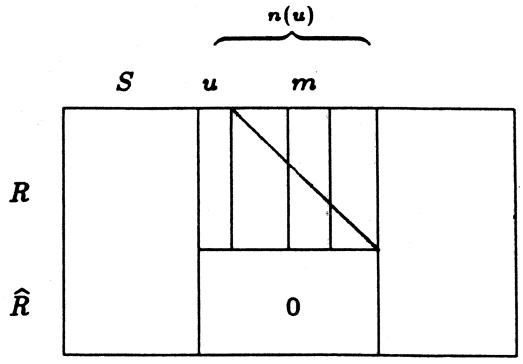

FIG. 4.1. *Proof of Theorem* 4.2: *Case* 2.

The triangular algorithm differs from GHM also in the strategy to select start columns. They use an initial QR factorization of A to identify a set of dependent columns, which is designated as the set of start columns. For each start column, they computed a null vector containing it, and any dependences not involving the start column were rejected to ensure that a basis was computed.

5. Computing triangular bases.

Implementation details. The initial matching $\mathcal{M}$ in the triangular algorithm is chosen by the column weighting strategy in § 3. The column u is chosen to be a column with most nonzeros in $U - S$, since once this null vector is computed, u will not be used again. The column m is chosen to be a column in $U - S$ rather than in M, if possible, since this saves the $O(\tau)$ operations needed to update the matching. The column v is chosen to be a column with fewest nonzeros in $U - S$ that can augment the matching.

In the modified circuit algorithm, the column c is chosen to be an inactive column of minimum cost. Here, the cost of a column c is the number of nonzeros it has in the inactive rows. Our heuristic justification is that this number of additional unmatched rows are added to a circuit when c is matched to an active row. In case of ties, two different tie breaking strategies were tried: columns with fewer total nonzeros were favored in one, and in the other, columns with most total nonzeros. The first strategy worked better for almost all problems.

Each null vector is computed by the triangular algorithm from a perfect matching in a submatrix of A. Each submatrix can have t rows and $t + 1$ columns, and hence the matching could cost $O(t\tau)$. The associated factorization could cost $O(t^3)$. The actual cost for each null vector should be lower since the number of rows in each submatrix should be small due to sparsity. Storage requirements for the algorithm are similar to that of the fundamental algorithm.

Results. An implementation of our algorithm in FORTRAN 77 forms the second part of BASIS. We present our results (computed under the same conditions as for fundamental null bases) in Table 4.

The triangular null bases we obtained are consistently sparser than fundamental bases. The increase in sparsity is most spectacular for *brandy* where the triangular basis has about half the density of the latter. In all cases, the densities of the constraint matrices and triangular null bases are comparable.

The time reported is the time (in seconds) needed to find a basis, given the matrix stored in Sparspak data structures and an initial complete matching. The triangular algorithm needs about two to five times the time needed by the fundamental algorithm. This is caused primarily by the $(n - t)$ matrix factorizations needed to compute the null vectors. The size of most of the matrices to be factored is quite small since the null vectors are sparse. Also, the LUSOL routines compute the sparse factorization efficiently. This explains why the increased time requirement is not greater.

The residuals in the system of equations from which null vectors are computed were always below machine precision. Condition numbers were estimated as before for the first eight problems in Table 2. These were all small, except for share1b and brandy (cond $(N) \approx 10^6$ for both); for the former the ratio cond (N)/cond (A) was about ten, and for the latter about one hundred.

Comparison with the Gilbert–Heath algorithms. We have also run the triangular algorithm on seven test problems shown in Table 5 from Gilbert and Heath (1987). Four of these are equilibrium matrices from structural optimization, and three, lp1, lp2 and lp3 are obtained from linear programs. The bases for the structural problems have a

TABLE 4
Triangular null bases.

	Null basis				Time (seconds)
Problem	Rows	Cols	Nonzeros	Density (%)	Total
murty	30	18	62	11.5	0.4
afiro	51	24	108	8.8	0.8
adlittle	138	82	486	4.3	3.6
share2b	162	66	686	6.4	5.0
share1b	253	136	1425	4.1	19.8
beaconfd	295	122	1581	4.4	70.4
israel	316	142	2118	4.7	34.4
brandy	303	110	2535	7.6	91.8
e226	472	249	2742	2.3	57.4
capri	482	211	2850	2.8	50.3
bandm	472	167	1941	2.5	26.4
stair	614	258	5094	3.2	59.9
etamacro	816	416	3563	1.0	59.5

natural profile structure arising from the locality of the interconnections in the physical structure, while the linear programs do not.

In Table 6, we compare the results from the triangular algorithm with results from the Gilbert and Heath matching algorithm (GHM) and turnback algorithm (GHT). Their code was executed on a reasonably similar setup to ours—a VAX 11/780 running the same operating system and on the same compiler. However, we need to be cautious about attaching too much significance to small differences in running times, since the algorithms are not being compared on the same machine.

The storage reported is the intermediate storage required to obtain a null vector. Two values are given for GHM and GHT. The dense storage reported is the maximum dense matrix storage needed for the active submatrix. The profile storage reported is the maximum storage that would be needed if a profile scheme is used to store the active submatrix. The running times pertain to an implementation that uses dense storage. The times (in seconds) in the Gilbert and Heath algorithms exclude the time needed for column pre-ordering and the initial QR factorization.

For the triangular algorithm, the storage reported is the maximum number of nonzeros in the L and U factors of the dependent set. We have not included the storage required by the integer arrays needed for column and row indices of nonzeros. This is quite small, since eight bytes are needed to store the nonzeros as double precision numbers, and only two bytes are needed to keep an integer.

TABLE 5
Test problems from Gilbert and Heath (1987).

Problem	Rows	Cols	Nonzeros	Density (%)
frame2d	27	45	93	7.7
lp1	57	97	465	8.4
frame3d	72	144	304	2.9
wheel	96	120	420	3.6
wrench	112	216	490	2.0
lp2	118	225	1182	4.5
lp3	171	320	906	1.7

TABLE 6
Results on problems from Gilbert and Heath (1987).

	Gilbert and Heath					Triangular Algorithm		
				Storage				
Problem	Algorithm	Nonzeros	Time	Dense	Profile	Nonzeros	Time	Storage
frame2d	matching	76	0.63	72	25	79	0.63	16
	turnback	76	2.30	288	27			
lp1	matching	367	10.50	1680	271	351	15.3	136
	turnback	391	28.02	2550	597			
frame3d	matching	317	2.95	168	41	310	3.6	35
	turnback	452	66.12	2064	104			
wheel	matching	488	10.45	756	273	518	5.5	103
	turnback	503	17.32	1560	326			
wrench	matching	518	26.98	3782	266	588	31.8	139
	turnback	544	58.38	5112	451			
lp2	matching	1363	103.68	8160	1784	1379	39.1	680
	turnback	1531	773.15	13570	8717			
lp3	matching	1101	60.80	3540	697	1210	78.9	169
	turnback	1518	288.87	10506	849			

The storage reported is the intermediate storage required to compute a null vector. For the Gilbert and Heath algorithms, the storage is the maximum size of the storage needed for the QR factors of the active submatrix. The dense storage refers to a dense matrix storage scheme, and the profile storage, to a profile matrix storage scheme. The reported times in GHM and GHT refer to the dense scheme. For the triangular algorithm, the storage is the maximum number of nonzeros in the sparse L and U factors of the dependent set.

Conclusions. The following conclusions may be drawn. Within the context of structural analysis problems, the use of turnback with profile storage scheme is justified. Because of the profile structure inherent in these null bases, the intermediate storage required is not prohibitive.

The two matching algorithms, GHM and the triangular algorithm, require smaller running times and less intermediate storage than turnback. The differences are greater for the linear programs, but this observation is true even for the structural problems. For instance, on lp2, GHT requires about twenty times the running time of the triangular algorithm, and more than twelve times the intermediate storage of the latter. We can conclude that a matching algorithm should be preferred over turnback for computing null bases of general sparse matrices.

Comparisons between GHM and the triangular algorithm are more difficult to make with the available data. Use of a profile scheme is essential to keep the intermediate storage from being prohibitive in GHM. However, the running times reported by Gilbert and Heath are for the dense storage scheme.

Both the algorithms compute fairly sparse null bases. The intermediate storage required by the GHM profile algorithm is of the same order as the storage required by the triangular algorithm. It is likely that the GHM profile algorithm will be a practical algorithm to compute sparse null bases.

We believe our results show clearly that the triangular algorithm is an attractive algorithm for large sparse null basis computations because of its low running times, small storage requirements, and the high sparsity achieved in the null bases.

6. Conclusions.

Sparse orthogonal null bases. In this section, we summarize some of our work on orthogonal null bases that is not included here.

We have shown that the circuit algorithm can be used to compute orthogonal null bases. By making use of the Dulmage–Mendelsohn decomposition of the square matched submatrix, we have also provided some theoretical evidence that such bases are unlikely to be sparse.

The computation of sparse orthogonal bases is further complicated by the fact that a greedy strategy may backfire; Pothen (1984) gives a counterexample to show that not choosing a sparsest null vector at a step can lead to a sparser orthogonal basis. In the nonorthogonal situation, a sparsest basis is always obtained by a greedy strategy.

We have designed algorithms to compute sparsest orthogonal bases for two special cases: a row vector of n elements, and a $t \times n$ dense matrix. For the vector, the sparsest basis has $n\lfloor \log_2 n \rfloor$ nonzeros, and for the matrix $nt\lfloor \log_2 n/t \rfloor$ nonzeros. These bases are computed by a recursive divide and conquer strategy. Proving that these bases are sparsest involves the solution of an interesting recurrence

$$f(n) = \min_{1 \leq k \leq n-1} f(k) + f(n-k) + n,$$

with $f(1) = 1$, and $f(2) = 4$. The reader can find the details in Pothen (1984) and Coleman and Pothen (1986b).

This algorithm has close connections with an algorithm to compute the orthogonal factorization on a distributed memory multiprocessor (Pothen, Jha and Vemulapati (1987)).

Summary. We have shown that matchings can be used to identify dependent sets of columns in a matrix, and thereby nonzeros in a null vector. These dependent sets are formed by choosing a start column and adding columns to it, one by one. Matchings help us in making good choices for columns to add so that a sparse null vector is obtained. This is accomplished by weighting columns and choosing a column of minimum weight to match to a row.

The resulting algorithms to compute null vectors have two phases: a combinatorial phase, in which dependent sets are identified, and a numeric phase, in which the coefficients of the null vector are computed. The time required for the second phase clearly dominates that of the first.

We have also focused attention on the structures of the null bases we construct. To ensure linear independence of the computed null vectors, the null bases are restricted to be triangular or fundamental.

To compute a fundamental basis, we need to ensure that the matched submatrix has full row rank. Only one sparse LU factorization of the matched submatrix and $(n - t)$ solves are needed to compute the basis. However, since all the null vectors are computed from a fixed initial matching, it is difficult to assign weights "globally" to columns in an intelligent way.

For each null vector in a triangular basis, we need to find a perfect matching in a submatrix, compute its LU factors, and perform a solve. In this case, a more intelligent dynamic column weighting strategy (of the modified circuit algorithm) is possible to ensure sparsity in each null vector.

Our computational results in Tables 3 and 4 demonstrate that both the fundamental and triangular algorithms succeed in computing sparse null bases, and require low running times and small intermediate storage. The triangular algorithm finds sparser bases at the expense of greater running times.

In Table 6, we compare the triangular algorithm with a turnback algorithm. The former identifies columns in a dependent set combinatorially, while the latter uses an orthogonal factorization. Consequently, the running times of the former are substantially lower. The triangular algorithm chooses columns to add to a dependent set by means of a combinatorial criterion to keep the set sparse, while turnback uses a numbering of the columns. Hence the triangular algorithm finds sparser bases. Finally, the triangular algorithm can use the column and row ordering schemes of a sparse LU factorization routine to keep intermediate storage low. The turnback algorithm cannot do so, since the columns in a dependent set are unknown before the completion of the orthogonal factorization.

For structural analysis problems with a natural profile structure in the bases, use of the turnback algorithm is justified. But for general sparse matrices, our results show that a combinatorial phase is essential to keep running times and storage low.

The conditioning of the null bases seems to depend on the conditions of the constraint matrices. When the latter were well-conditioned, the null bases were well-conditioned also, and there was less than a ten fold increase in estimated condition numbers. For one of the badly conditioned problems, the condition number increased a hundred fold. Developing algorithms that can control the conditioning of the null bases is an important open problem.

Additional remarks. Throughout this paper, we have assumed A has full row rank. The triangular algorithm can be modified to work in the rank deficient situation also. This can be done by rejecting unmatched rows in a maximum matching, since these are structurally dependent rows. The algorithm can then proceed until all unmatched columns have been used to compute null vectors. Then an LU factorization of M, the completely matched submatrix, can be used to identify the rest of the dependent columns.

Both the triangular and fundamental algorithms compute null vectors in a dependent set by computing LU factorizations; the LUSOL routines that do this have to decide when a column should be declared dependent in the factorization. This is a rather difficult numerical problem. Thus computing null bases is not immune from the difficulties associated with numerical rank determination.

The model of a circuit used by our algorithms is a submatrix with a complete matching which has one nonzero column more than its number of nonzero rows. This is appropriate when the matrix elements in A are "reasonably" random; hence, most circuits satisfy the weak Haar property.

When A is the vertex-edge incidence matrix of a directed graph, a circuit corresponds to a cycle in the graph, and a basis for the cycle space forms a null basis of A. A cycle has an equal number of nonzero columns and rows of the vertex-edge incidence matrix, unlike the circuits in this paper. Our algorithms will work correctly in this situation; however, sparser cycle bases could probably be obtained by a model that exploits this additional "structure".

Like vertex-edge incidence matrices of graphs, equilibrium matrices from structural analysis have additional structure. Most circuits have the number of nonzero rows greater than or equal to the number of nonzero columns. By considering "equilibrium graphs," bipartite graphs of equilibrium matrices, it is possible to exploit the structure in these problems, and to model circuits more accurately (Pothen (1986)). This approach yields a new algorithm to compute null bases for equilibrium matrices. This equilibrium graph algorithm succeeds in finding sparser bases faster than the triangular algorithm. In some cases even sparsest null bases can be characterized.

Computing sparse cycle bases is important in solving nonlinear programs with network constraints (Dembo (1983)). Little is known about computing sparsest cycle bases. Deo, Prabhu and Krishnamoorthy (1982) show that it is NP-complete to find sparsest fundamental cycle bases, and have designed heuristic algorithms (with implementations in Pascal) to find sparse fundamental bases.

Acknowledgments. Our thanks to John Gilbert and Mike Heath for sharing their experiences in designing their null basis code which helped us with our implementation; to Mike Saunders for sending the LUSOL package and the lp test problems to us and for being available with advice; and to the referees for their valuable suggestions to our paper.

Note added in proof. An $O(tt^3)$ algorithm has been designed to find the sparsest cycle basis of a graph by J. D. Horton (1987).

REFERENCES

M. W. BERRY, M. T. HEATH, I. KANEKO, M. LAWO, R. J. PLEMMONS AND R. C. WARD (1985), *An algorithm to compute a sparse basis of the null space*, Numer. Math., 47, pp. 483–504.

M. W. BERRY AND R. J. PLEMMONS (1985), *Computing a banded basis of the null space on the Denelcor HEP multiprocessor*, Proc. AMS/SIAM Summer Conference on Linear Algebra in Systems Theory, AMS Series on Contemp. Math.

J. A. BONDY AND U. S. R. MURTY (1976), *Graph Theory with Applications*, American Elsevier, New York.

THOMAS F. COLEMAN (1986), *On chordal preconditioners for large scale optimization*, Tech. Report 86-762, Computer Science Dept., Cornell Univ., Ithaca, NY.

THOMAS F. COLEMAN, ANDERS EDENBRANDT AND JOHN R. GILBERT (1986), *Predicting fill for sparse orthogonal factorization*, J. Assoc. Comput. Mach., 33, pp. 517–532.

THOMAS F. COLEMAN AND JORGE J. MORÉ (1983), *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20, pp. 187–209.

——— (1984), *Estimation of sparse Hessian matrices and graph coloring problems*, Math. Programming, 28, pp. 243–270.

THOMAS F. COLEMAN AND ALEX POTHEN (1986a), *The null space problem* I: *complexity*, this Journal, 7, pp. 527–537.

——— (1986b), *The null space problem* II: *algorithms*, Tech. Report 86-09, Computer Science, The Pennsylvania State University, University Park, PA; Tech. Report 86-747, Cornell Univ., Ithaca, NY.

RON S. DEMBO (1983), *A primal truncated Newton algorithm with application to large-scale network optimization*, Working Paper B-72, Yale School of Organization and Management, New Haven, CT.

NARSINGH DEO, G. M. PRABHU AND M. S. KRISHNAMOORTHY (1982), *Algorithms for generating fundamental cycles in a graph*, ACM Trans. Math. Software, 8, pp. 26–42.

I. S. DUFF (1977), MA28—*A set of Fortran subroutines for sparse unsymmetric linear equations*, AERE Report R8730, Harwell, England.

——— (1981), *On algorithms for obtaining a maximum transversal*, ACM Trans. Math. Software, 7, pp. 315–330.

DAVID GALE AND ALAN J. HOFFMAN (1982), *Two remarks on the Mendelson–Dulmage theorem*, Ann. Discrete Math., 15, pp. 171–177.

ALAN GEORGE AND JOSEPH W. LIU (1981), *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ.

JOHN R. GILBERT AND MICHAEL T. HEATH (1987), *Computing a sparse basis for the null space*, this Journal, 8 (1987), pp. 446–459.

P. E. GILL, W. MURRAY AND M. H. WRIGHT (1981), *Practical Optimization*, Academic Press, New York.

P. E. GILL, W. MURRAY, M. A. SAUNDERS AND M. H. WRIGHT (1986), *Maintaining LU factors of a general sparse matrix*, Tech. Report Systems Optimization Lab., Stanford Univ., Stanford, CA.

D. GOLDFARB AND S. MEHROTRA (1985), *A relaxed version of Karmarkar's method*, Tech. Report, Dept. I.E. & O.R., Columbia Univ., New York.

J. E. HOPCROFT AND R. M. KARP (1973), *A $n^{2.5}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput., 2, pp. 225–231.

J. D. HORTON, *A polynomial time algorithm to find the shortest cycle basis of a graph*, SIAM J. Comput., 16 (1987), pp. 358–366.

I. KANEKO, M. LAWO AND G. THIERAUF (1982), *On computational procedures for the force method*, Inter. J. Numer. Methods Engrg., 18, pp. 1469–1495.

EUGENE L. LAWLER (1976), *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.

BRUCE A. MURTAGH AND MICHAEL A. SAUNDERS (1983), *MINOS 5.0 user's guide*, Tech. Report SOL 83-20, Systems Optimization Lab., Stanford Univ., Stanford, CA.

KATTA G. MURTY (1983), *Linear Programming*, John Wiley, New York.

CHRISTOS H. PAPADIMITRIOU AND KENNETH STEIGLITZ (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, N.J.

ALEX POTHEN (1984), *Sparse null bases and marriage theorems*, Ph.D. thesis, Cornell Univ., Ithaca, New York.

——— (1986), *Equilibrium graphs in structural optimization*, Tech. Report 86-22, Computer Science Dept., Penn. State Univ., University Park, PA.

ALEX POTHEN, SOMESH JHA AND UDAYA VEMULAPATI (1987), *Orthogonal factorization on a distributed memory multiprocessor*, Hypercube Multiprocessors 1987, Michael T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 587–596.

J. K. REID (1976), *Fortran subroutines for handling sparse linear programming bases*, AERE Report R8269, Harwell, England.

——— (1982), *A sparsity-exploiting variant of the Bartels–Golub decomposition for linear programming bases*, Math. Programming, 24, pp. 55–69.

D. F. SHANNO AND R. E. MARSTEN (1985), *On implementing Karmarkar's method*, Working Paper 85-01, Dept. Man. Info. Systems, Univ. Arizona, Tempe, AZ.

M. N. THAPA (1984), *Optimization of unconstrained functions with sparse Hessian matrices*, Math. Programming, 29, pp. 156–186.

A. TOPCU (1979), *A contribution to the systematic analysis of finite element structures through the force method*, Ph.D. thesis, Univ. of Essen, Essen, Germany. (In German.)

P. WOLFE (1962), *The reduced gradient method*, The RAND Corporation, unpublished.