J. Opl Res. Soc., Pergamon Press, 1978. Vol. 29, 4, pp. 383 to 388. Printed in Great Britain

A Partitioned Inverse in Linear Programming

A. M. FRIEZE
Queen Mary College, London

This note deals with linear programs in which a subset of the constraints have a special structure. This structure allows linear equations involving these constraints only to be solved particularly easily, e.g. GUB rows. A method is described for restricting the gaussian elimination in LU decomposition to the non-special rows.

INTRODUCTION

GIVEN a large linear or integer linear programming problem one of the determinants of the problem's difficulty is the number of constraints. A large number of constraints will imply that the inverse basis matrix is of a large order. Sparsity will keep the actual amount of information needed to use the revised simplex method down to a fraction of what it could be.

A further reduction can be obtained if some of the constraints have a special structure. At the very least one would expect there to be a number of GUB rows, which can be handled within our framework.

For some problems, e.g. those associated with a particular integer programming formulation of plant location problems a technique similar to ours is essential if an integer programming solution is to be attempted at all.

This paper is concerned with linear programming problems of the form

$$\text{minimize } \mathbf{c}^T\mathbf{x}$$

subject to

$$\begin{aligned} A_1\mathbf{x} &= \mathbf{b}_1 \\ A_2\mathbf{x} &= \mathbf{b}_2, \end{aligned} \tag{1}$$

where $\mathbf{c}$, $\mathbf{x}$ are n-dimensional column vectors, $\mathbf{b}_i$ is an m_i-dimensional column vector for $i = 1,2$ and A_i is an $m_i \times n$ matrix of full row rank for $i = 1,2$.

The constraints have been partitioned to allow that A_1 is a matrix of a special structure which can be exploited in the revised simplex method. By a special structure we mean that for each non-singular $m_1 \times m_1$ submatrix D of A_1 that equations of the form

$$D\mathbf{x} = \mathbf{d} \quad \text{or} \quad D^T\mathbf{x} = d \tag{2}$$

can be solved relatively easily.

Examples

(a) *Generalized upper bounds* (see ref. 1). Here each variable x_j appears in at most one row of A_1 of the form

$$\sum_{j \in S_t} \alpha_j x_j = b_t,$$

where $S_1, S_2 \ldots$ are disjoint subsets of $\{1,2,\ldots,n\}$. Any non-singular submatrix D is diagonal and so (2) is trivial to solve.

(b) *Variable upper bounds* (see ref. 2). A variable upper bound VUB is defined by Schrage to be a constraint of the form

$$x_j \leqslant x_k \tag{3}$$

and it is assumed that a variable can appear either once as the LHS of a VUB or several times as the RHS of a VUB but not both. Such constraints appear in a wide variety of problems.[3–5] Adding a slack variable s_j and collecting these constraints to form A_1 we see that a non-singular submatrix D has the form

$$D = \begin{bmatrix} I_p & & & \\ & D_1 & & \\ & & \ddots & \\ & & & D_t \end{bmatrix},$$

where I_p is the $p \times p$ identity matrix and each D_i can be arranged in the form

$$\begin{bmatrix} 1 & & & -1 \\ & 1 & & -1 \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

It is clear that (2) is easy to solve in this case.

A very easy and natural generalization of this example considers constraints of the form

$$\sum_{j \in S_{ik}} a_j x_j \leqslant x_k \qquad \begin{matrix} i = 1, \ldots, n_k \\ k = 1, \ldots, p, \end{matrix}$$

where $S_{ik} \cap S_{jl} = \phi$ if $(i,k) \neq (j,l)$.

(c) *Distribution problems.* We assume that equations $A_1\mathbf{x} = \mathbf{b}_1$ are of the network flow variety and the constraints $A_2\mathbf{x} = \mathbf{b}_2$ represent some extra conditions.

It is shown by Dantzig[6] that a non-singular submatrix of A_1 can be represented by a spanning forest in the associated network and graph theoretic methods can be used to solve (2).

Many modern LP algorithms are based on the LU decomposition of Bartels and Golub[7]—see refs. 8 and 9 for improvements on their basic algorithm. Given the special structure assumed for A_1 it would seem advantageous to ignore the rows of A_1 when creating an LU decomposition in order to keep the special structure. This will generally lead to less "fill-in" and faster iterations possibly. We have not been able to test this hypothesis as yet although methods specifically designed for (a) and (b) above and different from our own, particularly (a), are accepted as being useful.

A SPECIALLY PARTITIONED INVERSE

Suppose that at some stage of applying the revised simplex algorithm an inversion is to be performed and that the basis matrix B has been partitioned

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \begin{matrix} \} m_1 \\ \} m_2 \end{matrix}$$
$$\qquad\quad \underbrace{\qquad}_{m_1} \quad \underbrace{\qquad}_{m_2}$$

so that B_{11} is non-singular. This partitioning is produced by the elimination procedure implied below.

Let

$$T = \begin{bmatrix} I_{m_1} & \\ -B_{21}B_{11}^{-1} & I_{m_2} \end{bmatrix}$$

then we have

$$TB = \begin{bmatrix} B_{11} & B_{12} \\ & \hat{B} \end{bmatrix},$$

where $\hat{B} = B_{22} - B_{21}B_{11}^{-1}B_{12}$. Next let

$$L = \begin{bmatrix} I_{m_1} & \\ & \hat{L} \end{bmatrix}$$

be such that

$$\hat{L}^{-1}\hat{B} = \hat{P}\hat{U}\hat{Q} = \hat{V},$$

where $\hat{P}$, $\hat{Q}$ are permutation matrices and $\hat{U}$ is upper triangular. The calculation of L is standard.[10]

Of the entities defined above one only needs to explicitly store the matrices $\hat{L}^{-1}$, $\hat{P}$, $\hat{Q}$ and $\hat{V}$ plus a knowledge of which columns are in B_{11} and which are in B_{12}.

At a general stage between inversions we will have a similar decomposition involving new matrices L, $\hat{V}$, P, Q and B and the original T such that

$$L^{-1}TB = \begin{bmatrix} B_{11} & B_{12} \\ & \hat{V} \end{bmatrix}.$$

To avoid confusion in the next section between the B_{11}, etc. in T and in B we will * the former.

SOLVING EQUATIONS USING THE DECOMPOSITION

(a) *Updating the column to enter the basis*

Let

$$\mathbf{a}_k = \begin{bmatrix} \mathbf{a}_{k1} \\ \mathbf{a}_{k2} \end{bmatrix}$$

be the column to enter the basis at a particular iteration then the updated column

$$\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}$$

is found by solving

$$B\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{a}_{k1} \\ \mathbf{a}_{k2} \end{bmatrix}$$

which is equivalent to

$$L^{-1}TB\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = L^{-1}T\begin{bmatrix} \mathbf{a}_{k1} \\ \mathbf{a}_{k2} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix}.$$

Now the computation of

$$\begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix}$$

is straightforward but involves computing

$$T\begin{bmatrix} \mathbf{a}_{k1} \\ \mathbf{a}_{k2} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_{k1} \\ \mathbf{a}_{k2} - B_{21}^{*}B_{11}^{*-1}\mathbf{a}_{k1} \end{bmatrix}.$$

The calculation of $B_{11}^{*-1}\mathbf{a}_{k1}$ is done by solving $B_{11}^{*}\mathbf{x} = \mathbf{a}_{k1}$ using the special structure.

One is left with solving

$$B_{11}\mathbf{y}_1 + B_{12}\mathbf{y}_2 = \mathbf{d}_1 \tag{3}$$

$$\hat{V}\mathbf{y}_2 = \mathbf{d}_2. \tag{4}$$

Equation (4) can be solved for $\mathbf{y}_2$ by back-substitution and then $\mathbf{y}_1$ computed using the special structure of B_{11}.

(b) *Calculating the reduced costs*

Suppose that $\mathbf{a}_k$ is to replace the pth column of B. Then letting $\mathbf{c}_B$ denote the vector of basic costs and $\mathbf{e}_p$ denote the pth row of $I_{m_1+m_2}$ we must compute $\mathbf{c}_B B^{-1}$ or $\mathbf{e}_p B^{-1}$. The latter vector can be used in a simple update of the simplex multipliers[11] and is needed in the steepest edge algorithms of Greenberg and Kaplan[12] or Goldfarb and Reid.[13]

Some computation can be solved in our method of updating the inverse representation if $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2) = \mathbf{e}_p B^{-1}$ is computed. We must therefore solve $\mathbf{u}B = \mathbf{e}_p$. If we put $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2) = \mathbf{u}T^{-1}L$ and $\mathbf{e}_p = (\mathbf{f}_1, \mathbf{f}_2)$ then the equations for v are

$$\mathbf{v}_1 B_{11} = \mathbf{f}_1 \tag{5}$$

$$\mathbf{v}_1 B_{12} + \mathbf{v}_2 \hat{V} = \mathbf{f}_2, \tag{6}$$

whose solution is similar to (3) and (4). The vector $\mathbf{u} = \mathbf{v}L^{-1}T$ can then be computed. (The post-multiplication by T will involve the solution of equations with coefficient matrix B_{11}^*.)

UPDATING THE INVERSE REPRESENTATION

Remark: let $\mathbf{y} = (y_1, \ldots, \mathbf{y}_m)$ be a column vector and $\mathbf{z} = (z_1, \ldots, z_m)$ be a row vector then the effect of pre-multiplying a matrix M by $I_m - \mathbf{y}\mathbf{z}$ is to subtract y_i times the vector $\mathbf{z}M$ from row i of M for $i = 1, \ldots, m$.

Now let p, m_1, m_2 be as before.

Case a—$p > m_1$.

Using primes to indicate the various quantities after a change of basis, we see that

$$L^{-1}TB' = \begin{bmatrix} B_{11} & B'_{12} \\ & S \end{bmatrix},$$

where S is a (row and column permuted) spiked matrix (see refs. 7–9). S may be transformed to a (row and column permuted) upper triangular matrix $\hat{V}'$ as in the above references. It should be noted that the spike is the column vector $\mathbf{d}_2$ obtained while updating the pivot column.

Case b—$p \leqslant m_1$.

We shall describe two methods for carrying out an update here.

The first method cannot always be expected to work with the Forrest–Tomlin update as it generally involves some fill-in in $\hat{V}$ before a spiked matrix is obtained. However, it should be borne in mind that a similar problem is faced in implementing the standard GUB procedure and Tomlin[14] states that it is not inefficient to re-invert instead whenever such fill-in is likely to occur.

For clarity assume now that $\hat{P} = \hat{Q} = I_{m_2}$ then we may express the product

$$L^{-1}TB' = \begin{bmatrix} B'_{11} & B_{12} \\ C & \hat{U} \end{bmatrix} \triangleq H, \tag{7}$$

when C is a matrix with one non-zero column. This being the vector $\mathbf{d}_2$ as in case a.

Method 1

Let $\mathbf{v}_1$ be as in (5) then it follows that $\mathbf{v}_1 B'_{11} = \lambda \mathbf{f}_1$ where λ may be zero. Now define $\mathbf{z} = (\mathbf{v}_1, 0)$—a row vector with $m_1 + m_2$ components the last m_2 being zero. Next

let $\mathbf{g} = (g_1, \ldots, g_{m_2}) = \mathbf{v}_1 B_{12}$ and consider the two possibilities:

(i) $\mathbf{g} = 0$

This implies $\lambda \neq 0$ else the new basis is singular and we define $\mathbf{y}$ to be a column vector with $m_1 + m_2$ components. The first m_1 are zero and the last m_2 components are equal to $\lambda^{-1}\mathbf{d}_2$.

(ii) $\mathbf{g} \neq 0$

Define t by $g_t \neq 0$ and $g_s = 0$ for $s < t$ and let $\mathbf{w}$ be the tth column of $\hat{U}$. Now define $\mathbf{y}$ with its first m_1 components zero and last m_2 components equal to $g_t^{-1}\mathbf{w}$.

Consider now the effect of pre-multiplying H in (7) by $I - \mathbf{yz}$. In (i) above this simply eliminates $\mathbf{d}_2$ from C and the update can be considered complete. In (ii) above this simply eliminates column t from $\hat{U}$ and alters the non-zero column of C if $\lambda \neq 0$. If we now interchange columns p, $m_1 + t$ of B' we are in effect faced with case a above and can complete accordingly.

It should be noted that the storage requirements for $I - \mathbf{yz}$ are simply the non-zeros of $\mathbf{y}$, $\mathbf{z}$ and calculating with $I - \mathbf{yz}$ is no more time consuming than with an ordinary eta vector of maximum length $m_1 + m_2$.

Method 2

We now consider how to avoid the drawback of producing the fill-in which the Forrest–Tomlin method was designed to avoid.

Let $\mathbf{g}$ be defined as above and if $\mathbf{g} = 0$ we can proceed as in method 1. If $\mathbf{g} \neq 0$ let t be defined as before. Let $\mathbf{e}_t$ be the tth row of I_{m_2} and let $\mathbf{w} = \mathbf{e}_t \hat{U}^{-1}$ which can be calculated by forward substitution. Now define $\mathbf{y}$ to be the $m_1 + m_2$ component column vector whose first m_1 components are zero, whose next $t - 1$ components are the first $t - 1$ components of column t of $\hat{U}$ and whose last $m_2 - t + 1$ components are zero. Next let $\mathbf{z}$ be the $m_1 + m_2$ component row vector whose first m_1 components are zero and whose last m_2 components are those of $\mathbf{w}$.

The effect of pre-multiplying H by $I - \mathbf{yz}$ is to delete the first $t - 1$ components of column t of $\hat{U}$ and perhaps alter the non-zero column of C. The tth diagonal element of $\hat{U}$ is then eliminated as in case b(ii). This may create some fill-in in row t of $\hat{U}$ but it is not disastrous. After interchanging columns p and $m_2 + t$ of B' we can still apply the $F - T$ algorithm because the fill-in in row t is then eliminated by the method.

A feature of the $F - T$ method is that the update of $\hat{U}$ goes on as the simplex multipliers are being updated and only one pass through $\hat{U}$ is needed. It seems that the update described here can also be done in one pass through $\hat{U}$ as the simplex multipliers are being updated.

PARTITIONING AND THE PRODUCT FORM OF INVERSE

We outline how one can proceed if one wishes to use a decomposition

$$L^{-1}TB = \begin{bmatrix} B_{11} & B_{12} \\ & I \end{bmatrix}. \tag{8}$$

We can use this expression to calculate reduced costs and update the pivot column, much as before. It is only really necessary to describe how to update this expression.

Case 1 $\mathrm{p} > \mathrm{m}_1$

The main effect on the right-hand side of (8) is that a column in I is replaced by an arbitrary column. The identity can be restored by premultiplying (8) by the corresponding eta vector.

Case 2 $\mathrm{p} \leqslant \mathrm{m}_1$

The main effect is to place an arbitrary vector $\mathbf{d}$ in the last m_2 rows of column p.

We will as in the previous section be in possession of a vector $\mathbf{u}$ such that $\mathbf{u}\bar{B}_{11} = \lambda \mathbf{f}_1$ and we define as before $\mathbf{g} = \mathbf{u}B_{12}$.

Case 2a $\mathbf{g} = 0$

Proceed as in method 1(i) of the previous section.

Case 2b $\mathbf{g} \neq 0$

Choose any index t such that $g_t \neq 0$ and then define $\mathbf{z} = (\mathbf{u}, 0)$ (an $m_1 + m_2$-vector) and $y = g_t^{-1}\mathbf{e}^r_{m_1+t}$. Pre-multiplying (8) by $I - \mathbf{yz}$ replaces the tth row of the unit matrix below B_{12} by $-g_t^{-1}(g_1, \ldots, 0, \ldots, g_{m_2})$ where 0 is in column t. We then use ordinary eta vectors to remove the non-zeros in this row, then interchange columns p and $m_1 + t$, which puts us back into case 1.

A NEW UPDATE FOR THE REVISED SIMPLEX ALGORITHM

A matrix $I - \mathbf{yz}$ can be constructed to return a spiked matrix S to upper triangularity without fill-in as follows: let U be upper triangular and suppose column p of U is replaced by $\mathbf{d}$ to create a spiked matrix S. Let U be $m \times m$ and let $\mathbf{e}_p$ be the pth row of I. Define $\mathbf{z} = \mathbf{e}_p U^{-1}$ and let $\lambda = \mathbf{zd}$ and define $\mathbf{y}$ by

$$\begin{aligned} y_j &= 0 && j = 1, \ldots p-1 \\ &= \lambda^{-1} d_j && j = p+1, \ldots m \\ &= 0 && j = p \text{ and } d_p \neq 0 \\ &= -1 && j = p \text{ and } d_p = 0. \end{aligned}$$

Then pre-multiplying S by $I - \mathbf{yz}$ removes spike components from below the diagonal and puts a 1 on the pth diagonal element if $d_p = 0$ thus maintaining non-singularity. Note that $\lambda \neq 0$ else $\mathbf{z}\,S = 0$.

REFERENCES

[1] G. B. DANTZIG and R. M. VAN SLYKE (1968) Generalised upper bounding techniques. *J. Comp. System Sci.* **1**, 213–226.

[2] L. SCHRAGE (1975) Implicit representation of variable upper bounds in linear programming. *Math. Prog. Study* No. 4, 118–132.

[3] M. A. EFROYMSON and T. L. RAY (1966) Branch-bound algorithm for plant location. *Ops Res.* **14**, 361–368.

[4] F. GLOVER and A. WOOLSEY (1974) Converting the 0–1 polynomial programming problem to a 0–1 linear program. *Ops Res.* **22**, 180–182.

[5] H. M. WAGNER and T. M. WHITIN (1958) Dynamic version of the economic lot size model. *Mgmt Sci.* **5**, 89–96.

[6] G. B. DANTZIG (1963) *Linear Programming and Extensions.* Princeton University Press, Princeton.

[7] R. H. BARTELS and G. H. GOLUB (1969) The simplex method of linear programming using LU decomposition. *Commun. Ass. Comput. Machinery* **12**, 266–268, 275–278.

[8] J. J. H. FORREST and J. A. TOMLIN (1972) Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Math. Prog.* **2**, 263–278.

[9] J. K. REID (1975) A sparsity exploiting variant of the Bartels–Golub decomposition for linear programming bases. A.E.R.E. Harwell Rep.

[10] J. A. TOMLIN (1972) Pivoting for size and sparsity in linear programming inversion routines. *J. IMA* **10**, 289–295.

[11] J. A. TOMLIN (1974) On pricing and backward transformation in linear programming. *Math. Prog.* **6**, 42–47.

[12] H. J. GREENBERG and J. E. KALAN (1975) An exact update for Harris's TREAD. *Math. Prog. Study* No. 4, 26–29.

[13] D. GOLDFARB and J. K. REID (1975) A practicable steepest-edge simplex algorithm. A.E.R.E. Harwell Rep.

[14] J. A. TOMLIN (1974) Generalised upper bounds and triangular decomposition in the simplex method. *Ops Res.* **22**, 664–668.