SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 617

January 1984

MODIFYING THE FORREST-TOMLIN AND SAUNDERS UPDATES FOR LINEAR PROGRAMMING PROBLEMS WITH VARIABLE UPPER BOUNDS

by

Michael J. Todd*

*Research supported in part by a fellowship from the Alfred P. Sloan Foundation and by NSF Grant ECS82-15361.

Typist: Anne T. Kline

Abstract

The author previously described a modification of the simplex method to handle variable upper bounds implicitly. This method can easily be used when the representation of the basis inverse (e.g. a triangular decomposition of the basis) is maintained as a dense matrix in core, but appears to cause difficulties for large problems where secondary storage and packed matrices may be employed. Here we show how the Forrest-Tomlin and Saunders updating schemes, which are designed for such large problems, can be adapted.

1. Introduction.

We are concerned with solution of the linear programming problem:

$$
(P) \qquad \begin{aligned} \min\ & c^T x \\ & Ax = h \\ & x_j \leq x_k, \qquad (k,j) \in E \\ & x \geq 0, \end{aligned}
$$

where the constraints $x_j \leq x_k$ for $(k,j) \in E$ are called variable upper bounds. Such problems arise frequently in applications, particularly as relaxations of integer programming formulations involving fixed charges. Examples are described by Schrage [11,12], Glover [6] and Todd [13], who also present special versions of the simplex method that exploit the structure of the variable upper bounds. We shall be concerned with the method proposed in [13], which circumvents the massive degeneracy caused by the special constraints and maintains a triangular decomposition of a small working basis.

If the basis is factored as the product of a lower and an upper triangular matrix, the updating steps are easy to perform if the latter is stored as a dense matrix in core. In this paper we address techniques to implement the method when the upper triangular factor is held in packed form. In particular, we show how the Forrest-Tomlin [5] and Saunders [10] schemes, with all their advantages when secondary storage is used, can be adapted to deal with this situation. Bastian [4] has shown how Forrest-Tomlin updating can be employed with Schrage's method [12] for generalized variable upper bounds; however, this method, in contrast to that in [13],

The iteration begins by solving $\pi^T B = d_B^T$ for the simplex multipliers π. Then π is used to price out the columns of A. This procedure is not completely standard, since it is designed to avoid degenerate pivots caused by the variable upper bounds. Thus we do not test for the entry of a nonbasic x_j whose father x_k is nonbasic. Instead, we compute reduced costs $\bar{c}_\ell = c_\ell - \pi^T a_\ell$ for $\ell = k$ and each son $\ell = j$ of k, and may choose to enter x_k together with all its "favorite sons" x_j, $j \in J_-(k)$, if $\bar{c}_k + \sum_{j \in J_-(k)} \bar{c}_j < 0$, where $J_-(k) = \{j \in J(k): \bar{c}_j < 0\}$. Full details are given in [13].

If optimality has not been achieved, we next construct a column b_q to enter the basis -- b_q is of the form $\pm a_j$, $j \in J$, or $a_k + \sum_{j \in J_-(k)} a_j$ -- and solve $Bz = b_q$ for z. The vector z allows us to make a minimum-ratio test to determine the next basic solution (or the existence of an unbounded solution). Again, this test is not standard, since it takes into account the variable upper bounds as well as the nonnegativity restrictions -- see [13].

Finally, we must update all quantities to allow the algorithm to proceed. Here we are particularly interested in the changes to the basis matrix B. Three cases are possible:

Case 1: The minimum ratio corresponds to a variable x_p hitting zero. In this case, the corresponding column, say b_p, of B is dropped and replaced by the new column b_q. Note that b_p may be (plus or minus) an original column of A or a composite column. This is a standard column exchange.

does not circumvent degeneracy with simple variable upper bounds. The rest of this section outlines the method of [13], with particular attention to the modification of the basis matrix during an iteration.

Suppose A is mxn, and let N denote $\{1,2,\ldots,n\}$. The simplest kind of variable upper bounds arise as follows. Let N be partitioned into J ("sons") and K ("fathers") and J be further partitioned into $J(k)$, $k \in K$. Suppose $(k,j) \in E$ iff $k \in K$ and $j \in J(k)$; that is, the "sons" of k (variables indexed by $j \in J(k)$) can be no greater than their "father". For simplicity, we shall usually consider this scenario, but extensions are possible: Glover [6] and Schrage [12] address quite general extensions, while Todd [13] allows the directed graph $G = (N,E)$ to be an out-forest, and allows "fatherless fathers" to have simple upper bounds.

The algorithm of [13] uses a working basis matrix B that is mxm, each of whose columns b_ℓ has the form $\pm a_j$, where $j \in J$, or $a_k + \sum_{j \in J_s(k)} a_j$, for some $J_s(k) \subseteq J(k)$, $k \in K$; here a_ℓ is the ℓth column of the matrix A. Basis columns of the latter kind (called composite columns) arise when the father x_k is positive, and certain of its sons, x_j, $j \in J_s(k)$, are equal to their upper bound x_k. The column takes its form from the fact that x_k and all x_j, $j \in J_s(k)$, move together. (This is not completely accurate: we may have $j \in J_s(k)$ and $0 < x_j < x_k$, in which case $-a_j$ is also a column of B, representing how far x_j is below x_k, but the reader may ignore such complications.) Corresponding to B is the cost vector d_B^T, which is constructed from c^T exactly as B is from A. We also have the updated right hand side vector $\bar{h} = B^{-1}h$, from which the values of the variables can easily be extracted.

Case 2: The minimum ratio corresponds to a variable x_p hitting its father x_k, where x_p is not the entering variable. In this case, the corresponding columns, say b_p and b_k, of B must be added to replace b_k. Then b_p is dropped and replaced by b_q. Note that we may alternatively add b_k to b_p and then drop b_k; the result is identical. Here we have a pre-processing step -- add one column of B to another -- followed by a column exchange, where the dropped column is one of the summands.

Case 3: The minimum ratio corresponds to the entering variable x_q hitting its father x_k. In this case we perform a standard column exchange, where the column of B corresponding to x_k, say b_k, is replaced by the new column $\hat{b}_k = b_k + b_q$.

(We have again slightly distorted the truth: for positive variables x_j with $j \in J_s(k)$ for some k, so that $-a_j$ is a column of B, the events "hitting zero" and "hitting its father" are reversed, but the changes to B are identical.)

These rules differ from the normal modifications to the basis matrix only in the possible pre-processing step in case 2. If B is factored as the product LU of a lower triangular and an upper triangular matrix (ignoring permutations), then this pre-processing corresponds to the addition of one column of U to another. Further, by choosing whether to add b_p to b_k or vice-versa, we can ensure that U remains upper triangular. Thus a simple modification of basis-updating routines suffices to handle this variant of the simplex method, as long as U is maintained as a dense matrix in core. However, it appears to be much more difficult to implement this method if U is held in packed form, possibly in secondary storage.

In his implementation of the Bartels-Golub basis factorization update [1,2], Reid [9] maintains U in packed form by rows, and also a column-oriented representation of the nonzeroes of U. Since each addition of, say, u_p to u_k is followed by dropping u_p, it is straightforward to update the row-oriented represention of U for a pre-processing step. If, in row i, u_{ip} were nonzero while u_{ik} were zero, then we merely update the pointer of the entry u_{ip} to indicate that it pertains to column k; otherwise, there is space to update u_{ik}. To update the column file, it seems necessary to write a new record for column k to correspond to its new sparsity pattern. Thus no great difficulties arise, though of course the representation of U is likely to be much denser and sometimes two new column records (instead of just one) need to be added during an iteration. This modification of Reid's routines will be tested in the future.

Reid's implementation is most suitable for an in-core code. For a large problem requiring secondary storage (e.g. disks), the basis-handling methods of Forrest and Tomlin [5] are much used. Here U is maintained in packed form by columns, and is read only, except for changing nonzero entries to zeroes. We show in the next sections how the Forrest-Tomlin updating scheme can be modified to handle pre-processing without changing its desirable features. Thus U is maintained in product form, with each factor modified only by replacing non-zeroes by zeroes or changing pointers. Factors are dropped less frequently than in the standard case, but many factors may be dropped simultaneously. Each iteration adds just one factor to the U^{-1}-file and one to the L^{-1}-file.

Saunders [10] uses a basis factorization scheme combining the numerical stability of the Bartels-Golub approach with the access advantages of Forrest and Tomlin. In the final section we sketch how Saunders' method can be adapted.

The extensions to variable upper bounds discussed in [13] can also be handled. Simple upper bounds on fathers change the logic of the method but the operations to be performed on the basis matrix are the same. Allowing an out-forest of variable upper bounds sometimes requires two column exchanges to be carried out, and also may entail adding one composite column of B to another, but otherwise involves no extra complication.

2. The Basis Representation.

Let B be an $m \times m$ nonsingular matrix, and let

$$L^{-1} B = U$$

$$L^{-1} = L_{n_L}^{-1} \ldots L_2^{-1} L_1^{-1} \tag{1}$$

$$U = U_{n_U} \ldots U_2 U_1$$

where each L_i^{-1} is an elementary row or column matrix (differing in just one row or column from the identity matrix) and each U_j is an elementary column matrix. Let $J = \{1,2,\ldots,n_U\}$. Suppose that we can write each U_j in the form

$$U_j = I + \tilde{u}_j f_j^T, \text{ where}$$

f_j is a unit vector, and $f_j^T \tilde{u}_i = 0$, $i < j$.

Since all cross-terms cancel, it follows that

$$U = I + \sum_{j \in J} \tilde{u}_j f_j^T .$$

Let e_i denote the ith unit vector in R^m, so that each f_j is some e_i. In fact, if $n_U > m$, each e_i may be represented several times - this is the essential new ingredient that allows us to handle pre-processing.

Pick $K \subseteq J$ with $|K| = m$ such that $\{f_k : k \in K\} = \{e_i : 1 \leq i \leq m\}$ and K consists of the "last representatives": $f_j = f_k$, $j \in J$, $k \in K$ imply $j \leq k$. Let J_k denote $\{j \in J : f_j = f_k\}$. (While the index sets J, K and J_k have different meanings from J, K and $J(k)$ in the introduction, there are strong parallels: J_k always corresponds to a father with some of its sons, for instance.)

To store U_j, we store $u_j = \tilde{u}_j$ if $j \notin K$ and $u_j = \tilde{u}_j + f_j$ if $j \in K$, together with (a pointer representing) f_j. Thus we write

$$\begin{aligned} U_j &= I + u_j f_j^T , && j \in J \backslash K \\ U_j &= I + (u_j - f_j) f_j^T , && j \in K, \text{ where} \\ & f_j \text{ is a unit vector and } f_j^T u_i = 0, \ i < j. \end{aligned} \tag{2}$$

Then

$$\begin{aligned} U &= U_{n_U} \cdots U_2 U_1 \\ &= I + \sum_{j \in J} \tilde{u}_j f_j^T \\ &= \sum_{j \in J} u_j f_j^T \\ &= \sum_{k \in K} \Big(\sum_{j \in J_k} u_j \Big) f_k^T , \end{aligned} \tag{3}$$

since $I = \sum_{k\in K} f_k f_k^T$. Let v_k denote $\sum_{j\in J_k} u_j$; then v_k is the column Uf_k of U.

Suppose $K = \{k_1,\ldots,k_m\}$ with $k_1 < \ldots < k_m$, and let P be a permutation matrix with $Pf_{k_i} = e_i$, $1 \le i \le m$. Let w_i denote Pv_{k_i} and $W = [w_1,\ldots,w_m]$. Then

$$PUP^T = \sum_{i=1}^{n} Pv_{k_i} (Pf_{k_i})^T$$

$$= \sum_{i=1}^{n} w_i e_i^T = W.$$

Moreover, if $i < j$ then $e_j^T w_i = f_{k_j}^T v_{k_i} = 0$. Thus W is upper triangular.

In the standard Forrest-Tomlin representation, $n_U = m$ and $J = K = \{1,2,\ldots,m\}$. In this case, the u_j's are the columns of U (in permuted order). Thus the representation (1)-(3) of the initial basis or of an intermediate or final basis at reinversion can be determined in a standard way, giving $n_U = m$. In particular, the preassigned pivot procedures of Hellerman and Rarick [7,8] can be used.

For updating, the special form of (2)-(3) will be most convenient, as we show below. Of course,

$$U_j^{-1} = I - \beta_j^{-1} u_j f_j^T \quad \text{with}$$

$$\beta_j = 1 + f_j^T u_j, \text{ for } j \in J\backslash K; \text{ and}$$

$$U_j^{-1} = I - \beta_j^{-1}(u_j - f_j) f_j^T \quad \text{with}$$

$$\beta_j = f_j^T u_j \quad \text{for } j \in K.$$

Thus it is trivial to obtain $U_j^{-1}x$ or $y^TU_j^{-1}$ in a number of operations a few more than the number of nonzeroes in u_j.

3. Pre-processing.

Assume that we have a representation as in (1)-(3) of the current basis matrix B. Note that each column of B, corresponding to a son variable of (P) or to a father together with some of its sons, corresponds to a column $v_k = Uf_k$ of U and hence to a set J_k of indices in J.

Suppose that we wish to add a column of B corresponding to a son to the column corresponding to its father (and some of its brothers, possibly). We therefore wish to add some column of U, say $v_p = Uf_p$, to another, say $v_k = Uf_k$. We may assume that $p, k \in K$ and that $p < k$ -- otherwise we interchange p and k. In this section, we show how to obtain an intermediate product-form representation of the resulting matrix

$$\bar{U} = U + ([v_p + v_k] - v_k)f_k^T + (f_p - v_p)f_p^T . \tag{4}$$

Thus in $\bar{U}$, the column v_k has been replaced by $v_p + v_k$ and the column v_p by f_p. We will subsequently replace the latter by a new column.

The new representation is given by quantities with overbars, $\bar{u}_j$, $\bar{f}_j$, etc. In one respect it is nonstandard, in that $|\bar{K}| = m - 1$ and no $\bar{f}_j = f_p$. However, this is rectified in the following column exchange.

Define

$$\begin{aligned} \bar{u}_j &= u_j,\ \bar{f}_j = f_j,\ j \in J\backslash J_p; \\ \bar{u}_j &= u_j,\ \bar{f}_j = f_k,\ j \in J_p. \end{aligned} \tag{5}$$

Then, since $p < k$, we again have $\bar{f}_j^T \bar{u}_i = 0$ for $i < j$. Let $\bar{K} = K\backslash\{p\}$. We have $\bar{J}_\ell = J_\ell$ for $\ell \in \bar{K}\backslash\{k\}$, while $\bar{J}_k = J_p \cup J_k$. Let

$$\begin{aligned} \bar{U}_j &= I + \bar{u}_j \bar{f}_j^T,\ j \in J\backslash\bar{K} \\ \bar{U}_j &= I + (\bar{u}_j - \bar{f}_j)\bar{f}_j^T,\ j \in \bar{K}. \end{aligned} \tag{6}$$

Then

$$\begin{aligned} \bar{U}_{n_U} \ldots \bar{U}_2 \bar{U}_1 &= I + \sum_{j \in J\backslash\bar{K}} \bar{u}_j \bar{f}_j^T + \sum_{j \in \bar{K}} (\bar{u}_j - \bar{f}_j)\bar{f}_j^T \\ &= \sum_{j \in J} \bar{u}_j \bar{f}_j^T + f_p f_p^T \\ &= \sum_{j \in J} u_j f_j^T + U f_p (f_k^T - f_p^T) + f_p f_p^T \\ &= \bar{U}, \end{aligned}$$

as desired. Note that we only have to change pointers: $\bar{f}_j$ becomes f_k for $j \in J_p$.

As a final remark, note that, with the simplest kind of variable upper bounds, either J_p or J_k -- whichever corresponds to the son variable -- is a singleton, while the other, corresponding to the father, might have several elements. With the extensions to the concept mentioned in the

introduction neither J_p nor J_k need be singletons. The modifications described above make no assumptions on the forms of J_p or J_k.

4. Column Exchange.

Suppose now that we wish to replace an old column of B (and of U) by a new one. We assume that we have either a representation (1)-(3) of U or an intermediate representaton of the result $\overline{U}$ of a pre-processing step as in the previous section.

The modification of the representation consists of two steps: the old column is first replaced by a unit column, and then the new column replaces this. The first step depends slightly on whether pre-processing was just performed.

4.1 Dropping a column, no pre-processing.

Suppose that we wish to drop the column $v_p = Uf_p$ of U and replace it with f_p, to get

$$\begin{aligned} \tilde{U} &= U + (f_p - v_p)f_p^T \\ &= \sum_{j\in J\setminus J_p} u_j f_j^T + f_p f_p^T \end{aligned} \tag{7}$$

It is clear that we only need to eliminate the factors U_j for $j \in J_p$ to get a product-form representation for $\tilde{U}$. However, to allow the subsequent insertion of the new vector, we change this representation using an elementary row operation R.

Define $w^T = f_p^T U^{-1}$ and $\lambda = f_p^T v_p = f_p^T Uf_p$. Since U is permuted upper triangular, $\lambda = (f_p^T U^{-1} f_p)^{-1} = (w^T f_p)^{-1}$. Define $R = I - f_p(\lambda w^T - f_p^T)$, so that

$$R^{-1} = I + f_p(\lambda w^T - f_p^T). \tag{8}$$

(In the case where $n_U = m$, R reduces to Forrest and Tomlin's R (and to Bastian's [3] R^{-1}) and w^T is similar to Forrest and Tomlin's r^T and equals Bastian's w^T.)

Then set

$$\begin{aligned}
\overset{v}{U} &= R^{-1}\tilde{U} \\
&= (I - f_p f_p^T + \lambda f_p w^T)\,\tilde{U} \\
&= \textstyle\sum_{j\in J\setminus J_p}[(I - f_p f_p^T)u_j]f_j^T + \lambda f_p w^T\tilde{U} \\
&= \textstyle\sum_{j\in J\setminus J_p}[(I - f_p f_p^T)u_j]f_j^T + \lambda f_p f_p^T(1 + w^T(f_p - v_p)) \\
&= \textstyle\sum_{j\in J\setminus J_p}[(I - f_p f_p^T)u_j]f_j^T + f_p f_p^T ,
\end{aligned} \tag{9}$$

since $w^T v_p = f_p^T U^{-1} U f_p = 1$ and $w^T f_p = \lambda^{-1}$. Thus $\overset{v}{U}$ has the product-form representation where each transformation U_j, $j \in J_p$, is deleted, and u_j is replaced by $\overset{v}{u}_j = (I - f_p f_p^T)u_j$ (eliminating its entry $f_p^T u_j$, if present) for $j \in J\setminus J_p$. Note that $f_p^T \overset{v}{u}_j = 0$ for all undeleted j; this allows the insertion of the new vector. (The extra term $f_p f_p^T$ is caused by the fact that no undeleted f_j equals f_p.)

4.2 Dropping a column after pre-processing.

Now we have a product form representation of $\overline{U}$, and $\overline{U}f_p$ is already the unit column f_p. As above, define $w^T = f_p^T \overline{U}^{-1}$ and $\lambda = f_p^T \overline{U} f_p = (w^T f_p)^{-1} = 1$.

With

$$R^{-1} = I + f_p(w^T - f_p^T) \tag{10}$$

we have

$$\begin{aligned} \overset{v}{U} &= R^{-1}\overline{U} \\ &= (I - f_p f_p^T + f_p w^T)\overline{U} \\ &= (I - f_p f_p^T)(\textstyle\sum_{j\in J}\overline{u}_j\overline{f}_j^T + f_p f_p^T) + f_p f_p^T \\ &= \textstyle\sum_{j\in J}[(I - f_p f_p^T)\overline{u}_j]\overline{f}_j^T + f_p f_p^T . \end{aligned} \tag{11}$$

Thus $\overset{v}{U}$ has the product-form representation where each $\overline{u}_j$ is replaced by $\overset{v}{u}_j = (I - f_p f_p^T)\overline{u}_j$ by eliminating its entry $f_p^T u_j$, if present. Again we have $f_p^T \overset{v}{u}_j = 0$ for all j. In this case, no transformations $\overline{U}_j$ are deleted.

4.3 Adding a column.

Finally we wish to insert a new column, say b, into the basis. Let the basis matrix be $\hat{B}$, and let $u = R^{-1}L^{-1}b$. Then

$$\begin{aligned} R^{-1}L^{-1}\hat{B} &= \overset{v}{U} + (u - f_p)f_p^T \\ &= \hat{U}, \end{aligned} \tag{12}$$

say. Then $\hat{U} = \sum_{j\in J} \overset{v}{u}_j\overline{f}_j^T + uf_p^T$, or $\sum_{j\in J\setminus J_p} \overset{v}{u}_j f_j^T + uf_p^T$ according

as pre-processing was, or was not, performed. Thus we obtain a representation (1)-(3) for the new basis $\hat{B}$ by defining:

$$\hat{L}_j^{-1} = L_j^{-1},\ 1 \le j \le n_L;$$

$$\hat{L}_{n_L+1} = R^{-1}; \tag{13}$$

$$\hat{u}_j = \overset{v}{u}_j,\ \hat{f}_j = f_j,\ j \in J\backslash J_p \quad \text{(no processing)}$$

or

$$\hat{u}_j = \overset{v}{u}_j,\ \hat{f}_j = \overline{f}_j,\ j \in J \quad \text{(after processing)}$$

$$\hat{u}_{n_U+1} = u,\ \hat{f}_{n_U+1} = f_p.$$

5. Implementation.

Here we show how the updating method of the previous two sections can be implemented efficiently, minimizing access to the L^{-1}- and U^{-1}-files. As in the standard Forrest-Tomlin update, we postpone the updating of L^{-1} and U^{-1} till the beginning of the next iteration, when the simplex multipliers π^T are computed. Thus at the end of the iteration, we have determined the type of update required and we have available, as a byproduct of the computation of $B^{-1}b_q$, the vector $L^{-1}b_q$.

We now run through the U^{-1}-file, simultaneously computing w^T, updating, and computing $d_{\tilde{B}}^T \hat{U}^{-1}$. We may also need to extract v_p.

Let us say "Ci is true" if the iteration falls into Case i of the introduction, for i = 1,2,3. Thus C1 implies a standard column exchange, C2 requires preprocessing, and C3 forces us to extract v_p.

Initialize: set $w^T \leftarrow f_p^T$, $\pi^T \leftarrow d_{\hat{B}}^T$, $v \leftarrow 0$, $f \leftarrow f_p$, $P \leftarrow I - f_p f_p^T$.

do $j = 1,2,\ldots,p-1$

if $j \notin J_p$ then $\pi^T \leftarrow \pi^T U_j^{-1}$;

else if C2 then $f_j \leftarrow f_k$

$\pi^T \leftarrow \pi^T U_j^{-1}$;

else mark j deleted

if C3 then $v \leftarrow v + u_j$;

end.

if C2 then $f_p \leftarrow f_k$

$w^T \leftarrow w^T U_p^{-1}$

$u_p \leftarrow Pu_p$

$\pi^T \leftarrow \pi^T U_p^{-1}$

$\lambda \leftarrow 1$;

else $w^T \leftarrow w^T U_p^{-1}$

mark p deleted

$\lambda \leftarrow f_p^T u_p$

if C3 then $v \leftarrow v + u_p$.

do $j = p + 1,\ldots,n_U$

$w^T \leftarrow w^T U_j^{-1}$

$u_j \leftarrow Pu_j$

$\pi^T \leftarrow \pi^T U_j^{-1}$

end.

$u \leftarrow L^{-1} b_q + v.$

Form R^{-1} from w^T and $f(=$ old $f_p)$ as in (8).

$n_L \leftarrow n_L + 1.$

$n_U \leftarrow n_U + 1.$

$L_{n_L}^{-1} = R^{-1}.$

$u_{n_U} \leftarrow R^{-1}u.$

$f_{n_U} \leftarrow f.$

$\pi^T \leftarrow \pi^T U_{n_U}^{-1}$.

Now we have updated the basis inverse representation: we continue by completing the computation of π^T by setting $\pi^T \leftarrow \pi^T L_{n_L}^{-1}\ldots L_1^{-1}$. Of course, a few extra pointers are required, to indicate the index set K, the status of each variable, and so on, and these must also be updated in the obvious way.

In summary, we have shown how the files containing the basis inverse representation can be updated in the next BTRAN operation, and that these files essentially need be read only, as in the original Forrest-Tomlin scheme.

6. The modification of Saunders' basis-handling scheme.

In [10], Saunders proposes that the upper triangular matrix in a basis decomposition be permuted to the form

$$W = \begin{pmatrix} D & R \\ 0 & G \end{pmatrix}$$

where D is diagonal, R is essentially read only, and G is a small matrix which can be maintained in primary storage, in packed form. Here we sketch how the ideas of the preceding sections can be employed to adapt Saunders' method to the requirements of preprocessing.

Write

$$W = W_I W_{II} = \begin{pmatrix} D & 0 \\ 0 & G \end{pmatrix} \begin{pmatrix} I & D^{-1}R \\ 0 & I \end{pmatrix}$$

If $PUP^T = W$, write $PU_iP^T = W_i$, $i = I, II$. Then $U = U_I U_{II}$. We maintain U_I (or just D and a representation of G) in core, while U_{II} is kept in a product-form representation as in (2)-(3) in secondary storage. In fact, it is convenient to use the $\tilde{u}_j$'s of section 2 rather than the u_j's to represent each factor U_j - in other words, the identity diagonal of U_{II} is ignored. No factors are necessary for the first block of columns in W_{II}.

We now describe how each type of update is performed. The vector b is the new column of B. We concentrate on the product form of U_{II}, on D and on G, assuming the latter is stored as a dense matrix; details of updating permutations and storing G in packed form are omitted.

6.1 No pre-processing, column of $\begin{pmatrix} R \\ G \end{pmatrix}$ dropped.

Delete the corresponding factor(s) in the product form of U_{II}. Append a new factor whose $\tilde{u}_j$ is D^{-1} times the appropriate subvector of $L^{-1}b$, padded with zeroes. Drop the corresponding column of G and add the new column (the rest of $L^{-1}b$); reduce to upper triangular form with a Bartels-Golub update.

6.2 No pre-processing, column of $\begin{pmatrix} D \\ 0 \end{pmatrix}$ dropped.

Extract and delete the corresponding row of $D^{-1}R$, say $d_i^{-1}r_i^T$. Append a new factor as above (with the ith entry removed). Remove d_i from the representation of D. Form the new matrix $\begin{bmatrix} r_i^T & u_i \\ G & \hat{u} \end{bmatrix}$ where $\begin{pmatrix} u_i \\ \hat{u} \end{pmatrix}$ denotes the appropriate part of $L^{-1}b$. Reduce to upper triangular form with a Bartels-Golub update.

6. 3 Pre-processing, adding one column of $\begin{pmatrix} R \\ G \end{pmatrix}$ to another.

Update the product form of U_{II} by first changing f_p's to f_k as in section 4 and then appending a new factor with $f_j = f_p$. Add the appropriate column, say g_p, to the other, say g_k, drop g_p, add the new column and proceed with a Bartels-Golub update.

6.4 Pre-processing, adding a column of $\begin{pmatrix} D \\ 0 \end{pmatrix}$ to one of $\begin{pmatrix} R \\ G \end{pmatrix}$.

Extract and delete the corresponding row of $D^{-1}R$, say $d_p^{-1}r_p^T$. Append a new factor as in 6.2. Form the new matrix $\begin{bmatrix} r_p^T + d_p e_k^T & u_p \\ G & \hat{u} \end{bmatrix}$ as in 6.2, and reduce to upper-triangular form. Remove d_p from the representation of D.

6.5 Pre-processing, adding one column of $\begin{pmatrix} D \\ 0 \end{pmatrix}$ to another.

Extract and delete both corresponding rows of $D^{-1}R$, say $d_p^{-1}r_p^T$ and $d_k^{-1}r_k^T$. Append a new factor as in 6.2 (missing pth and kth components and "pivoting" in the pth row). Form the new matrix

$$\begin{bmatrix} d_p & r_p^T & u_p \\ d_k & r_k^T & u_k \\ 0 & G & \hat{u} \end{bmatrix}$$

and reduce to upper triangular form with a Bartels-Golub update. Remove d_p and d_k from the representation of D.

Except for 6.3, where one column of G must be added to another, and 6.5, where G increases both dimensions by two, these steps are identical to normal updates in Saunders' method. Moreover, case 6.5 can only occur when both "father" and "son" belong to the initial part of W; since updating causes composite columns to belong to the second part of W, this can only occur with columns placed in the initial part of W during a refactorization, and hence will typically happen infrequently.

7. Conclusion.

The basis-handling schemes of Forrest-Tomlin and Saunders can be modified to deal with the requirements of a specialized algorithm for linear programming problems with variable upper bounds. These modifications preserve the desirable features of the original methods; in particular, most of the basis inverse representation can be held in secondary storage and accessed sequentially.

References

[1] R.H. Bartels, "A stabilization of the simplex method", Numerische Mathematik 16 (1971) 414-434.

[2] R.H. Bartels and G.H. Golub, "The simplex method for linear programming using LU decomposition", Communications of the ACM 12 (1969) 266-268.

[3] M. Bastian, "Updated triangular factors of the working basis in Winkler's decomposition approach," Mathematical Programming 17 (1979) 391-397.

[4] M. Bastian, "Implicit representation of generalized variable upper bounds using the elimination form of the inverse on secondary storage," to appear in Mathematical Programming.

[5] J.J.H. Forest and J.A. Tomlin, "Updating triangular factors of the basis to maintain sparsity in the product form of the simplex method", Mathematical Programming 2 (1972) 263-278.

[6] F. Glover, "Compact LP bases for a class of IP problems", Mathematical Programming 12 (1977) 102-109.

[7] E. Hellerman and D. Rarick, "Reinversion with the preassigned pivot procedure", Mathematical Programming 6 (1971) 195-246.

[8] E. Hellerman and D. Rarick, "The partitioned preassigned pivot procedures (P4)", in Sparse Matrices and their Applications, ed. by D.J. Rose and R.A. Willoughby, Plenum Press, New York (1972) 67-76.

[9] J.K. Reid, "A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases," Mathematical Programming 24 (1982) 55-69.

[10] M.A. Saunders, "A fast stable implementation of the simplex method using Bartels-Golub updating", in Sparse Matrix Computations, ed. by J.R. Bunch and D.J. Rose, Academic Press, New York, (1976) 213-226.

[11] L. Schrage, "Implicit representation of variable upper bounds in linear programming", Mathematical Programming Study 4 (1975) 118-132.

[12] L. Schrage, "Implicit representation of generalized variable upper bounds in linear programming", Mathematical Programming 14 (1978) 11-20.

[13] M.J. Todd, "An implementation of the simplex method for linear programming problems with variable upper bounds," Mathematical Programming 23 (1982) 34-49.