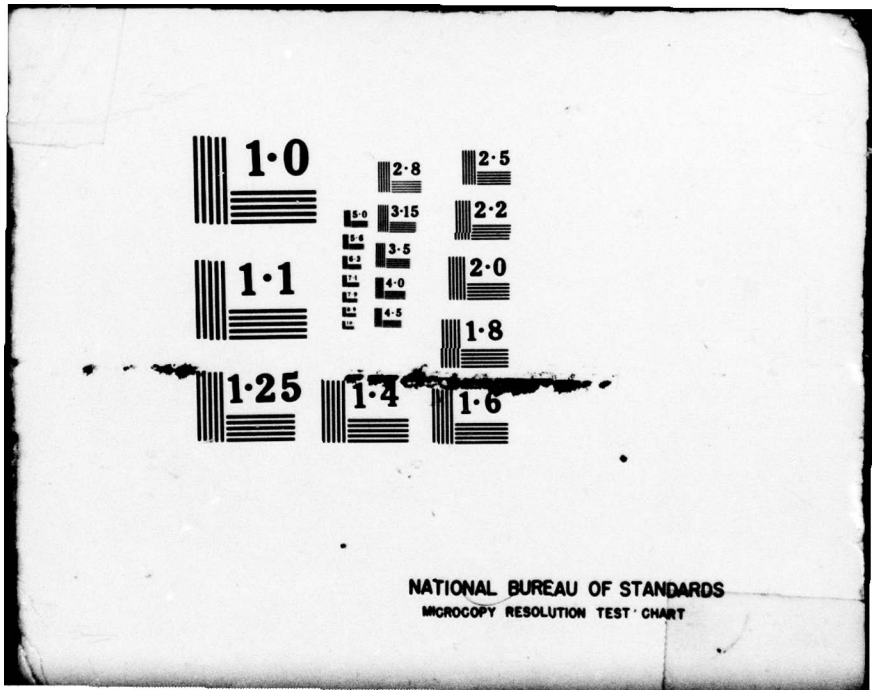
AD-A060 976 STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB F/G 12/1
A BASIS FACTORIZATION METHOD FOR BLOCK TRIANGULAR LINEAR PROGRA--ETC(U)
APR 78 A F PEROLD, G B DANTZIG N00014-75-C-0267
UNCLASSIFIED SOL-78-7 NL

1 OF 2
ADA
060976



1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

Systems
Optimization
Laboratory

LEVEL II

12

AD A060976

DDC FILE COPY

DDC
NOV 7 1978
F

This document has been approved for public release and sale; its distribution is unlimited.

Department of Operations Research
Stanford University
Stanford, CA 94305

78 11 02 039

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
Stanford University
Stanford, California
94305

A BASIS FACTORIZATION METHOD FOR BLOCK TRIANGULAR LINEAR PROGRAMS

by

Andre F. Perold and George B. Dantzig

TECHNICAL REPORT SOL 78-7
April 1978

Research and reproduction of this report were partially supported by the Office of Naval Research Contract N00014-75-C-0267; the Department of Energy Contract EY-76-S-03-0326-PA-#18; the National Science Foundation Grants MCS76-81259, MCS76-20019, and ENG77-06761.

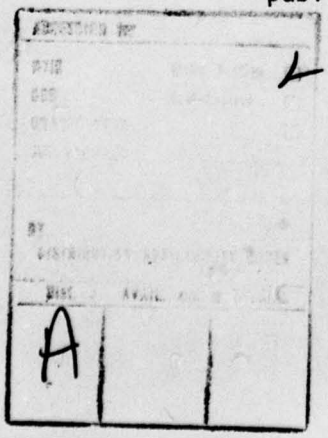


TABLE OF CONTENTS

Section		Page
1	Introduction.	1
2	Persistence in Staircase Models	6
3	The Square Block Triangular Factorization of the Basis.	12
4	Minimal Factorizations.	15
5	Solving Linear Equations.	23
	5.1. Solving Against General Right Hand Sides.	23
	5.2. Solving Against Structured Right Hand Sides . . .	29
6	Updating the Factorization.	35
	6.1. Review of Updating an Inverse Representation of a Matrix When an Exchange of Columns has Taken Place	36
	6.2. Updating $\bar{B}$	37
	6.3. Updating G	40
	6.4. Discussion and Summary of the Fundamental Updating Operations	49
7	Implementation.	54
	7.1. Factorizing and Updating the $\bar{B}_{tt}$	55
	7.2. Factorizing and Updating G	57
	7.3. Recomputing the Factorization $B = \bar{B}F$	62
	7.4. Looking at the Fundamental Operations	64
8	Estimating the Work per Simplex Iteration Accounted for by the Factorization.	67
	8.1. Q Stored in Product form Versus Q Stored Explicitly	67
	8.2. The Contribution of $\bar{B}$.	72
	8.3. Summary	72
9	Computational Results and a Further Analysis.	74
	9.1. Analysis of the Number of Multiplications per Iteration	76
	9.2. Different Strategies During Update.	81
	9.3. A Comparison with MINOS	84
	9.4. Storage Considerations.	87
	9.5. Bumps and Spikes.	89
10	Conclusion.	93
	Acknowledgments	95
	References.	96

1. Introduction

Consider the linear programming problem

$$\text{minimize} \quad c_1^T x_1 + \cdots + c_K^T x_K$$

$$\text{subject to} \quad A_{11} x_1 = b_1$$

$$A_{s1} x_1 + \cdots + A_{ss} x_s = b_s \qquad s = 2, \ldots, K$$

$$x_t \geq 0 \qquad t = 1, \ldots, K$$

where

x_t is a vector of n_t variables

A_{st}, $s \geq t$ is an $m_s \times n_t$ matrix

c_t and b_s are vectors of dimension n_t and m_s respectively,

$s,t = 1, \ldots, K$.

Let

$$n = \sum_{t=1}^{K} n_t, \qquad m = \sum_{s=1}^{K} m_s \ .$$

Such linear programs have a block triangular structure, as can be readily observed from the $m \times n$ detached coefficient matrix

$$A = \begin{bmatrix} A_{11} & & & \\ A_{21} & A_{22} & & \\ \vdots & & & \\ A_{K1} & A_{K2} & \cdots & A_{KK} \end{bmatrix}$$

Row i and col j of A are said to belong to period t if a_{ij} is an element of A_{tt}, where $A = (a_{ij})$.

This is an important class of problems: it includes all multi-staged and time-staged linear programs that arise either naturally as, for example, dynamic economic models of investment and planning in the energy sector, or, as discretizations of continuous time linear control problems (Dantzig [7]). Often, sparse general large scale linear programs can be permuted to block triangular form.

Linear programs of this type are difficult to solve using standard simplex method techniques because they usually require a disproportionately large number of iterations (Beale [1]). As a result, several attempts have been made to devise algorithms that exploit this structure.

As far back as 1955 (Dantzig [5]) a key empirical observation was made: <u>for dynamic (time-staged) models similar type activities are likely to persist in the basis for several periods</u>. To take advantage of this possibility an algorithm for block triangular systems

was outlined. The central idea was to use a square block triangular 'artificial basis', that is, one with square blocks on the diagonal, together with a factor to correct for the off-square diagonal blocks of the true basis. In essence this correction factor would differ from the $m \times m$ identity matrix in as many columns as the true basis was off-square on the diagonal. The implication of the persistence observation was that these columns would be small in number and so require little work in being maintained from one iteration to the next. Further, to solve equations with the square block triangular factor, only the diagonal blocks need by factorized. This could yield a substantial savings in storage requirements, as well as an increase in speed.

In this paper we shall extend these ideas by examining the persistence property in detail, by characterizing factorizations in which the correction factor has the minimal number of structural columns, and by showing that only a small submatrix of the correction factor need be maintained to execute the simplex algorithm. Then we shall develop in detail the methods of updating the factorization. In the remaining sections we shall discuss the implementational details, perform an analysis of the work involved, and present some computational experience.

In [14] Kallio and Porteus also employ a square block triangular factorization along the lines suggested in [5]. However their factorization and methods of updating differ substantially from ours. The

factorization we consider always maintains the square block triangular factor as a submatrix of A; in [14], this factor contains some transformed columns, making it denser than ours and more difficult to store, but yielding a sparser correction factor. No computational experience is reported in [14].

Other methods devised specifically for this class of problems have considered mainly staircase structures, that is, where the matrices $A_{st} = 0$ for $t < s - 1$. See for example the nested decomposition algorithm of Ho and Manne [12] and the block factorization techniques of Loute [16] and Wollmer [29]. McBride [17] uses a dynamic factorization of the basis after inducing a block triangular structure with the use of Hellerman and Rarick's P^4 algorithm [11]. He makes no prior structural assumptions and allows the block triangular partition to vary from one iteration to the next.

Bisschop and Meeraus [3] have proposed a method for very general L.P.'s that is not based on any triangularization or block triangularization techniques, but uses an idea that to some extent underlies the method presented here as well as that of [17]: If we wish to solve the nonsingular system $Dy = d$, but have at our disposal a nonsingular factorized matrix E, where E differs from D in (say) k columns, then there exists a suitable $k \times k$ nonsingular matrix P such that the solution to $Dy = d$ can be obtained by solving systems involving only E and P. The method presented in [3] uses this idea to save substantially on storage costs over the conventional LU methods, by confining the growth of nonzeros to P.

We shall present empirical results that point to the factorization of this paper being superior to both the above methods (i.e., [3], [17]).

2. Persistence in Staircase Models

Time-staged staircase linear programs are often discrete versions of an underlying continuous process that evolves as the solution to a differential equation. One such continuous model is the following linear optimal control problem with mixed constraints on the state and control variables:

$$\text{minimize} \quad \int_0^T \{c^T x(t) + d^T u(t)\}dt + q^T x(T)$$

$$\text{subject to} \quad \left.\begin{aligned} \dot{x} &= Ax + Bu + a \\ 0 &= Cx + Du + b \end{aligned}\right\} \ \ell \ \text{equations}$$

$$x(t) \geq 0 \qquad u(t) \geq 0 \qquad t \in [0,T] \qquad x(0) = p \ .$$

It has been shown in [22] that if $(x(\cdot), u(\cdot))$ is a nondegenerate extreme point solution of the above (convex) constraint set, and if $u(\cdot)$ is piecewise continuous, then the interval $[0,T]$ can be partitioned into a collection of open intervals (possibly an infinite number) such that on each interval, precisely ℓ components of (x,u) are strictly positive, and the remaining ones are identically zero.

In such cases we would hope that as we make finer and finer the discrete approximation to the continuous-time problem, the basic solutions will tend to behave more and more in this way, i.e., have the

same activities basic over several consecutive time periods, and moreover have precisely ℓ of them do so. That this is often the case in practice was observed by Dantzig in [5].

In the following we shall investigate this persistence property by looking at the structural properties of bases in general staircase models where each step has the same size, say $\ell \times r$. Such models have constraints of the form

$$A_1 x_1 = b_1$$

$$B_{t-1} x_{t-1} + A_t x_t = b_t, \qquad t = 2, \ldots, K$$

$$x_t \geq 0$$

where the coefficient matrices A_t and B_t are all $\ell \times r$. To any basis B will correspond matrices P_t and Q_t that are submatrices of A_t and B_t respectively, so that

$$B = \begin{bmatrix} P_1 & & & \\ Q_1 & P_2 & & \\ & Q_2 & P_3 & \\ & & \ddots & \\ & & & Q_{K-1} \; P_K \end{bmatrix}$$

P_t and Q_t will have the same number of columns (which can vary from one t to the next); for all t, P_t and Q_t will have ℓ rows. For $t = 1, \ldots, K$ let k_t be such that P_t and Q_t are $\ell \times (\ell + k_t)$. Clearly k_t may be of any sign and is a measure of the "off-squareness" of period t in this basis. Note that since B is square,

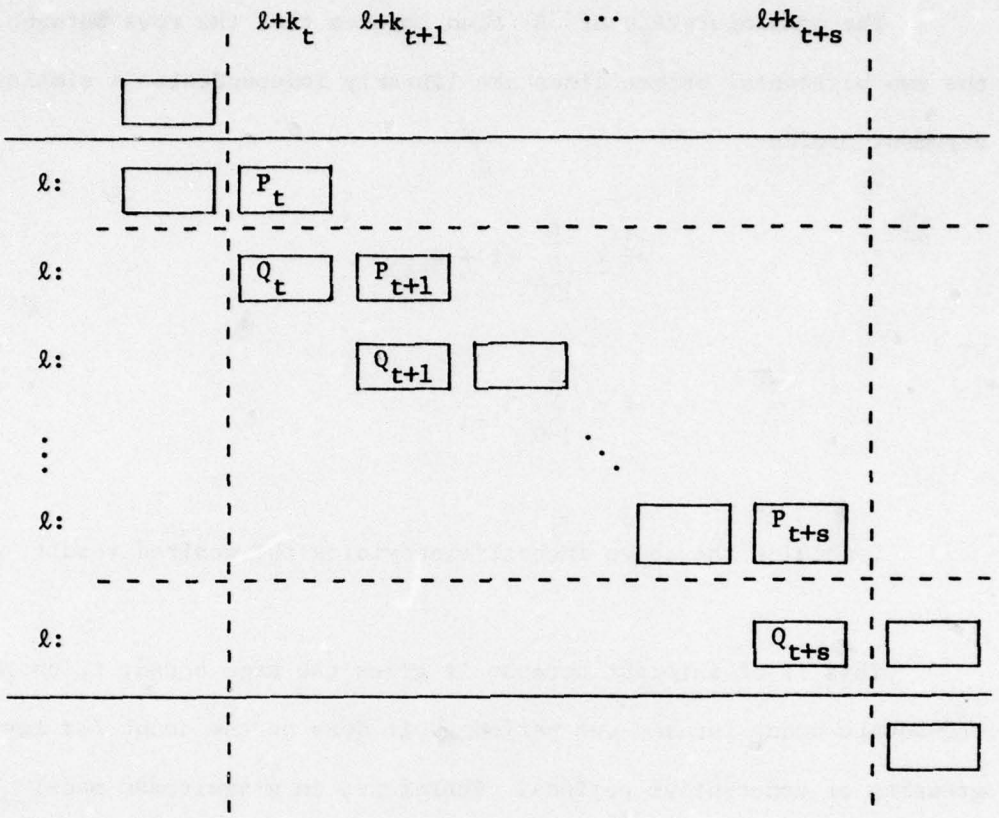
$$\sum_{t=1}^{K} k_t = 0 .$$

The following theorem gives bounds on how far off-square the basis can be along the diagonal.

2.1. Theorem: For all t and s such that $1 \leq t \leq t + s \leq K$,

$$-\ell \leq \sum_{j=0}^{s} k_{t+j} \leq \ell$$

Proof: Consider the portion of B between t and $t + s$:



Since B is nonsingular, the columns between the two vertical broken lines are linearly independent. Therefore, since these columns have only zeros outside the two horizontal solid lines, the number of these columns may not exceed the number of rows between the two solid lines, i.e.,

$$\sum_{j=0}^{s} (\ell + k_{t+j}) \leq (s + 2)\ell$$

or

$$\sum_{j=0}^{s} k_{t+j} \leq \ell$$

The nonsingularity of B also implies that the rows between the two horizontal broken lines are linearly independent. A similar argument yields

$$s\ell \leq \sum_{j=0}^{s} (\ell + k_{t+j})$$

or

$$-\ell \leq \sum_{j=0}^{s} k_{t+j} .$$

Combining the above inequalities yields the desired result. □

This is of interest because it gives the same bound, ℓ, on the off-square count for any one period as it does on the count for any grouping of consecutive periods. Therefore, in a staircase model where the number of time periods, K, is very large relative to ℓ, we can choose to group together, say, every s periods to obtain a coarser partition of the matrix in such a way that the ratio of the off-square count to the number of periods in any partition, ℓ/s, is small. Indeed, as K becomes arbitrarily large and the number of partitions remains constant, the off-square ratio for any partition becomes arbitrarily small.

In many economic applications we can attach a stronger although more qualitative significance to the result of Theorem 2.1. Activities like the level of coal production, or the level of an inventory, are

usually positive over intervals of time, rather than at points spread haphazardly. In continuous-time this corresponds to a statement about the piecewise continuity of optimal solutions. In discrete-time, we infer that such activities will remain basic over a whole time interval, irrespective of how many time steps constitute the interval. The implication of this for the $\{k_t\}$ is that they will be constant over intervals of time no matter how refined the grid size. Setting $k_{t+j} = k$ for $j = 0, 1, \ldots, s$ in the theorem and noting that k is an integer, we obtain $k = 0$ for grid sizes refined enough, that $s \geq \ell$. Thus in solutions where the activities persist in the basis over intervals of time, and each such interval contains at least $\ell + 1$ time steps, the number of activities in each time step is precisely ℓ. Note though, that this will not hold true at the end points of the time intervals.

In time-staged models where the matrix structure is block triangular but not staircase, the above discussion and Theorem 1.1 do not apply. However, in cases where the non-staircase part of the block triangular structure contains just a sprinkling of nonzeros, and similar type activities can appear from one period to the next, it seems reasonable to expect a similar behavior.

3. The Square Block Triangular Factorization of the Basis

Let B denote the $m \times m$ basis matrix. Since B is a submatrix of A, it too will have a block triangular structure

$$B = \begin{bmatrix} B_{11} & & & \\ B_{21} & B_{22} & & \\ \vdots & & & \\ B_{K1} & B_{K2} & \cdots & B_{KK} \end{bmatrix}$$

Note that the diagonal blocks B_{tt} need not be square in general, and that it is possible that there be no contribution to B from one of the 'periods', i.e., for some i, the submatrix

$$\begin{bmatrix} 0 \\ B_{tt} \\ \vdots \\ B_{Kt} \end{bmatrix}$$

may be vacuous. Note also that since B is nonsingular, it is necessary that the leading submatrices

$$B_{11}, \begin{bmatrix} B_{11} & \\ B_{21} & B_{22} \end{bmatrix}, \cdots$$

have full row rank and hence have at least as many columns as rows. Likewise the submatrices

$$B_{KK}, \begin{bmatrix} B_{K-1,K-1} & \\ B_{K,K-1} & B_{K,K} \end{bmatrix}, \cdots$$

all have full column rank and hence have at least as many rows as columns. However, very little else may be said about B in general.

We shall factorize B into the product of two $m \times m$ nonsingular matrices[1]

$$B = \bar{B} F \tag{1}$$

where

$$\bar{B} = \begin{bmatrix} \bar{B}_{11} & & & \\ \bar{B}_{21} & \bar{B}_{22} & & \\ \vdots & & \ddots & \\ \bar{B}_{K1} & \bar{B}_{K2} & \cdots & \bar{B}_{KK} \end{bmatrix}$$

[1] Dantzig [5] works with the (mathematically) equivalent factorization $\bar{B} = BE$ where E is F^{-1}.

with the diagonal blocks $\bar{B}_{tt}$ square and nonsingular. $\bar{B}$ will be called the <u>artificial</u> basis.

To understand the structure of F, consider the case when the i^{th} column of B, b_i, is equal to the j^{th} column of $\bar{B}$, $\bar{b}_j$. Letting e_j denote the j^{th} column of the $m \times m$ identity matrix, it is clear that

$$b_i = \bar{B}\, e_j$$

so that the i^{th} column of F must be e_j. Thus if $\bar{B}$ and B have k columns in common F will contain k unit columns and $m - k$ other columns. By suitably permuting the rows and columns of F we can obtain

$$F = P \begin{bmatrix} G & \\ H & I \end{bmatrix} Q \qquad (2)$$

where P and Q are permution matrices, G is $(m - k) \times (m - k)$ and nonsingular, and I is the identity matrix of crder k.

4. Minimal Factorizations

As we shall emphasize later the usefulness of this factorization approach will depend critically on the number of columns that B and $\bar{B}$ have in common, or, more directly, the size of the matrix G in (2). It will be important to know how to obtain a G that is as small as possible.

4.1. Definition: Given a basis B, a square block triangular factorization of B, $B = \bar{B}F$, will be said to be minimal if for any other such factorization, $B = \bar{B}'F'$, $\bar{B}$ has at least as many columns in common with B as does $\bar{B}'$.

4.2. Notation: For two matrices C and D with the same number of rows, let $C \cap D$ denote the submatrix of columns that are in both C and D.

Thus a factorization $\bar{B}F$ is minimal if (number of columns of $B \cap \bar{B}$) $\geq$ (number of columns of $B \cap \bar{B}'$) for all admissable $\bar{B}'$.

We have the following useful characterization of minimal factorizations.

4.3. Theorem: Let B be a given basis. Then a square block triangular factorization of B, $B = \bar{B}F$, is minimal if and only if for $t = 1, \ldots, K$, $B_{tt} \cap \bar{B}_{tt}$ is a basis for the space spanned by the columns of B_{tt}.

Proof: Suppose there is a t such that $B_{tt} \cap \bar{B}_{tt}$ is not a basis for the column space of B_{tt}. Then there is a column b of B,

$$b = \begin{bmatrix} 0 \\ b_t \\ \vdots \\ b_K \end{bmatrix}$$

such that the augmented matrix

$$W = [B_{tt} \cap \bar{B}_{tt} \vdots b_t]$$

has full column rank.

Since $\bar{B}_{tt}$ is nonsingular, b_t has a representation

$$\bar{B}_{tt}\, y = b_t \,. \tag{3}$$

Since W has full column rank there is a column $\bar{b}$ of $\bar{B}$,

$$\bar{b} = \begin{bmatrix} 0 \\ \bar{b}_t \\ \vdots \\ \bar{b}_K \end{bmatrix}$$

such that $\bar{b}_t$ is a column of $\bar{B}_{tt}$ but not of B_{tt}, and such that the coefficient of $\bar{b}_t$ in the representation of b_t in (3) is nonzero. It follows that the matrix $\bar{B}'_{tt}$ arrived at by interchanging columns $\bar{b}_t$ and b_t is nonsingular. Since $\bar{b}_t$ is not in B_{tt}, $\bar{b}$ is not in B. Let $\bar{B}'$ be $\bar{B}$ with the columns b and $\bar{b}$ interchanged. Then $\bar{B}'$ is again square block triangular with nonsingular diagonal blocks. Further, $\bar{B}'$ contains one more column of B than did $\bar{B}$. Therefore the factorization $\bar{B}F$ is not minimal.

Conversely if the factorization $\bar{B}F$ is not minimal there is a factorization $\bar{B}'F'$ such that $B \cap \bar{B}'$ has more columns than $B \cap \bar{B}$. Thus for some t, $B_{tt} \cap \bar{B}'_{tt}$ has more columns than $B_{tt} \cap \bar{B}_{tt}$. Also, since $\bar{B}'_{tt}$ is nonsingular $B_{tt} \cap \bar{B}'_{tt}$ has full column rank. Hence $B_{tt} \cap \bar{B}_{tt}$ cannot be a basis for the column space of B_{tt}.

This completes the proof. □

4.4. Applications of Theorem 4.3 : Theorem 4.3 serves two useful purposes. Firstly, it provides us with a constructive method for computing a minimal $\bar{B}$ and F for any B: For each t, find a linearly independent subset, M_{tt}, of columns of B_{tt} that is as large as possible. Augment M_{tt} with any other convenient columns, e.g., unit columns, to form a square and nonsingular matrix $[M_{tt} \vdots N_{tt}]$, say. Then let $\bar{B}$ consist of the columns that correspond to the M_{tt}, together with the N_{tt}, suitably padded with zeros. $\bar{B}$ will thus have $[M_{tt} \vdots N_{tt}]$ as its diagonal blocks. The Theorem tells us that this is a minimal $\bar{B}$.

Secondly, from the proof of the Theorem, we have a means of constructing a minimal factorization from any given square block triangular factorization, $B = \bar{B}F$, as follows. Suppose that the j^{th} column of B, say b_j, is not in $\bar{B}$. For some t we can write

$$b_j = \begin{bmatrix} 0 \\ b_t \\ \vdots \\ b_K \end{bmatrix} .$$

The j^{th} column of F say f_j, is the representation of b in terms of $\bar{B}$. Since b_j has zero in the first $t - 1$ periods, it follows by square block triangularity of $\bar{B}$, that f_j has the form

$$f_j = \begin{bmatrix} 0 \\ f_t \\ \vdots \\ f_K \end{bmatrix} .$$

Thus

$$b_t = \bar{B}_{tt} f_t .$$

and if f_t has a nonzero component corresponding to any column that is in $\bar{B}$ but not in B, we can replace this column by b_j, and so obtain a 'better' factorization. Repeating this must yield a minimal factorization.

Remark: As a final remark regarding the structure of F in relation (2), note that G is simply that submatrix of F whose columns are indexed by the columns of B that are not in $\bar{B}$, and whose rows are indexed by those columns of $\bar{B}$ that are not in B.

We shall say that a column (row) of G is in period t if the corresponding column of $B(\bar{B})$ is in period t.

From the above, it is then clear that a factorization $\bar{B}F$ is minimal iff whenever a column of G is in period t (some t), it has zeros in all the rows in period t.

To conclude this section we shall illustrate with the following 3 period example.

4.5. Example:

$$B = \begin{array}{|cc|cc|c|} \hline 1 & 1 & & & \\ \hline 2 & 0 & 1 & 2 & \\ 3 & 1 & 0 & 0 & \\ \hline 4 & 0 & 1 & 1 & 1 \\ 5 & 0 & 0 & 0 & 1 \\ \hline \end{array}$$

To construct a minimal factorization we find

$$M_{11} = [1] \qquad M_{22} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad M_{33} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

M_{11} is already square and invertible, thus requiring no augmentation. However, we augment M_{22} and M_{33} with unit vectors to yield

$$\begin{bmatrix} M_{22} & N_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} M_{33} & N_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} .$$

Thus

$$\bar{B} = \begin{array}{|c|cc|cc|} \hline 1 & & & & \\ \hline 2 & 1 & 0 & & \\ 3 & 0 & 1 & & \\ \hline 4 & 1 & 0 & 1 & 0 \\ 5 & 0 & 0 & 1 & 1 \\ \hline \end{array} .$$

To find F we solve $\bar{B}F = B$, one column at a time, to yield

$$F = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & -2 & 1 & 2 & 0 \\ 0 & -2 & 0 & 0 & 0 \\ 0 & -2 & 0 & -1 & 1 \\ 0 & -3 & 0 & 1 & 0 \end{bmatrix}$$

Note that columns 1, 3, 5 of B are columns 1, 2, 4 of $\bar{B}$ respectively, and their columns 1, 3, 5 of F are unit vectors with ones in rows 1, 2, 4.

Also columns 3 and 5 of $\bar{B}$ are not in B and columns 2 and 4 of B are not in $\bar{B}$. The elements in rows 3 and 5 and in columns 2 and 4 of F yield

$$G = \begin{bmatrix} -2 & 0 \\ -3 & 1 \end{bmatrix} .$$

The rows of G correspond to columns 3 and 5 of $\bar{B}$ which in turn are in periods 2 and 3 respectively, while the columns of G correspond to columns 2 and 4 of B which in turn are in periods 1 and 2 respectively, as indicated below.

			Period	Row of F
$G =$	-2	0	2	3
	-3	1	3	5

Period	1	2
Col of F	2	4

For a factorization to be minimal it is necessary and sufficient that $g_{ij} = 0$ if row i and column j belong to the same period, where $G = (g_{ij})$. In this case row 1 and column 2 both belong to period 2, and $g_{12} = 0$. Therefore the factorization is minimal.

5. Solving Linear Equations

Two major steps in the revised simplex method (Beale [2]) are to solve the equations

$$By = a \tag{4}$$

where y is the representation of the entering column, a, and

$$B^T \pi = c \tag{5}$$

where π is the vector of prices used to determine the next entering column.

5.1. Solving Against General Right Hand Sides

Using the factorization $B = \bar{B}F$ to solve for example, the system $By = a$, we would, respectively, solve the systems

$$\bar{B}z = a$$

$$Fy = z \quad .$$

The system $\bar{B}z = a$ is easy to solve since $\bar{B}$ is square block triangular. Partition z and a to conform with the partition of $\bar{B}$:

$$z = \begin{bmatrix} z_1 \\ \vdots \\ z_t \\ \vdots \\ z_K \end{bmatrix} \qquad a = \begin{bmatrix} a_1 \\ \vdots \\ a_t \\ \vdots \\ a_K \end{bmatrix} .$$

We would solve respectively

$$\begin{aligned} \bar{B}_{11}\, z_1 &= a_1 \\ \bar{B}_{22}\, z_2 &= a_2 - \bar{B}_{21}\, z_1 \\ &\vdots \\ \bar{B}_{KK}\, z_k &= a_K - \bar{B}_{K1}\, z_1 - \bar{B}_{K2}\, z_2 - \cdots - \bar{B}_{K,K-1}\, z_{K-1} \end{aligned} \tag{6}$$

As we shall show later on, it will always be possible to have $\bar{B}$ be a submatrix of A, the original detached coefficient matrix. Thus, finding the terms $B_{st}\, z_s$, $s > t$, in the above right hand side would usually involve little work, since the matrices B_{st} are usually very sparse, and in fact often zero.

The system $Fy = z$, however, is more troublesome. From (2) we must solve

$$P \begin{bmatrix} G & \\ H & I \end{bmatrix} Qy = z \quad .$$

Both G and H are computed matrices, and are typically between 20% and 40% dense. Further, if G is $k \times k$, then H is $(m - k) \times k$, and for m much larger than k, the storage requirements for H and the work involved in maintaining H could become prohibitive.

In the following we shall show however that it is possible to solve the system $By = a$ without knowledge of H.

To simplify matters we shall assume that we have suitably reordered the columns of B to be BQ^T and the columns of $\bar{B}$ to be $\bar{B}P$, and refer to these reordered matrices as B and $\bar{B}$, so that

$$B = \bar{B} \begin{pmatrix} G & \\ H & I \end{pmatrix} \quad .$$

(For now we are disregarding the original block triangular structure we had an B and $\bar{B}$.) Partitioning B and $\bar{B}$ according to the partition of F, we may write

$$B = (B^1 \quad B^2) \qquad \bar{B} = (\bar{B}^1 \quad \bar{B}^2)$$

where

$$B^2 = \bar{B}^2 = B \cap \bar{B} \quad .$$

Thus

$$[B^1 \quad B^2] = [\bar{B}^1 \quad \bar{B}^2] \begin{bmatrix} G & \\ H & I \end{bmatrix} \quad . \tag{7}$$

Note that

$$\begin{bmatrix} G \\ H \end{bmatrix} = \bar{B}^{-1} B^1 \quad . \tag{8}$$

To solve $By = a$, partition

$$y = \begin{bmatrix} y^1 \\ y^2 \end{bmatrix}$$

so that

$$[B^1 \quad B^2] \begin{bmatrix} y^1 \\ y^2 \end{bmatrix} = a \tag{9}$$

Since $B^2 = \bar{B}^2$ this is equivalent to solving

$$B^1 y^1 + [\bar{B}^1 \quad \bar{B}^2] \begin{bmatrix} 0 \\ y^2 \end{bmatrix} = a \quad . \tag{10}$$

Multiplying (10) by $\bar{B}^{-1}$, and using (5) we get

$$\begin{bmatrix} G \\ H \end{bmatrix} y^1 + \begin{bmatrix} 0 \\ y^2 \end{bmatrix} = \begin{bmatrix} z^1 \\ z^2 \end{bmatrix} \tag{11}$$

where

$$z = \begin{bmatrix} z^1 \\ z^2 \end{bmatrix}$$

satisfies

$$\bar{B}z = a \quad .$$

From (11), y^1 must satisfy the relation

$$G\, y^1 = z^1 \tag{12}$$

y^2 may then be obtained by solving (10) with y^1 determined in (12).

In short, to solve $By = a$, we solve three sets of equations

$$\left.\begin{aligned} \bar{B}z &= a \\ G\,y^1 &= z^1 \\ \bar{B}w &= a - B^1 y^1 \end{aligned}\right\} \qquad (13)$$

where w is

$$\begin{bmatrix} 0 \\ y^2 \end{bmatrix} .$$

There is a similar procedure for the system $B^T \pi = c$, again with three sets of equations:

$$\left.\begin{aligned} \bar{B}^T \lambda &= \begin{bmatrix} 0 \\ c^2 \end{bmatrix} \\ G^T \mu &= c^1 - (B^1)^T \lambda \\ \bar{B}^T \pi &= \begin{bmatrix} \mu \\ c^2 \end{bmatrix} \end{aligned}\right\} \qquad (14)$$

To verify this notice that the first and last relations in (14) yield

$$B^T \lambda = F^T \bar{B}^T \lambda = F^T \begin{bmatrix} 0 \\ c^2 \end{bmatrix} = \begin{bmatrix} H^T c^2 \\ c^2 \end{bmatrix}$$

and

$$B^T \pi = F^T \bar{B}^T \pi = F^T \begin{bmatrix} \mu \\ c^2 \end{bmatrix} = \begin{bmatrix} G^T \mu + H^T c^2 \\ c^2 \end{bmatrix} .$$

Therefore the second relation in (14) yields

$$G^T \mu + H^T c^2 = c^1$$

so that

$$B^T \pi = \begin{bmatrix} c^1 \\ c^2 \end{bmatrix}$$

as required.

In sum, the equations (13) and (14) show that in order to solve the systems (4) and (5) we need only be able to solve systems involving $\bar{B}$ and G. This means that to execute the simplex method the storage and running costs are now localized to the factorizations of the K diagonal blocks of $\bar{B}$ as well as the factorization of G (that is, apart from the pricing and minimum ratio operations).

5.2. Solving Against Structured Right Hand Sides

Section 4.1 dealt with the case when the right hand sides of equations (4) and (5) are general vectors. However, most of the time, these right hand sides are structured, and taking advantage of this can cut down considerably on the work involved.

(i) In equation (4) the right hand side, a, is almost always an incoming column. Thus if the column a is from period t, it will have the form

$$a = \begin{bmatrix} 0 \\ a_t \\ \vdots \\ a_K \end{bmatrix}$$

and to solve the first equation in (13) i.e., $\bar{B}z = a$, we need only solve the reduced system

$$\begin{aligned} \bar{B}_{tt}\, z_t &= a_t \\ \bar{B}_{t+1,t+1}\, z_{t+1} &= a_{t+1} - \bar{B}_{t+1,t}\, z_t \\ &\vdots \\ \bar{B}_{KK}\, z_K &= a_K - \bar{B}_{K,1}\, z_1 - \cdots - \bar{B}_{K,K-1}\, z_{K-1} \end{aligned} \tag{15}$$

since we know that

$$z_1 = 0, \ldots, z_{t-1} = 0 \ .$$

On the average, we would then solve $[\frac{K}{2}]$ small systems, where $[\cdot]$ denotes "the smallest integer greater than." Note that this would not apply to the third step in (13) since the right hand side there has no structure in general. Nevertheless the total number of small systems to be solved in (4) is now on the average $K + [\frac{K}{2}]$ for $\bar{B}$ and 1 for G.

The only time when we are not able to take advantage of the structure of a is when solving for the current basic solution

$$Bx = b$$

where b is the right hand side of the original L.P. formulation. However, this is only done once every, say, 50 - 100 iterations.

(ii) In equation (5) the right hand side, c, is always during Phase II, the same unit vector

$$e_\ell = (0 \cdots \underset{\substack{\uparrow \\ \ell^{th}}}{1} \cdots 0)^T$$ [1]

where the objective row appears in row ℓ of the A matrix. In modern implementations of the simplex method (Beale [2]), this will

[1] We have assumed here that the cost row $(c_1, c_2, \ldots, c_K)$ in the initial formulation has been imbedded as one of the rows of the A matrix, as is customary.

not be the case during Phase I since c is set equal to a vector of -1's, 0's and 1's depending on which variables are currently outside their bounds.

For the case $c = e_\ell$ we shall show that by suitably enlarging G, we can solve for μ in (14) without first computing λ. Recall that from the derivation of equations (14) we had

$$F^T \begin{bmatrix} \mu \\ c^2 \end{bmatrix} = \begin{bmatrix} c^1 \\ c^2 \end{bmatrix} .$$

However with $c = e_\ell$, $\begin{bmatrix} \mu \\ c^2 \end{bmatrix}$ is simply the ℓ^{th} column of $(F^T)^{-1}$. Now since the objective variable associated with the ℓ^{th} row of A, the cost row, is unrestricted in sign, the basis B will always contain its column, which is also the unit column e_ℓ. Further we can always ensure that the factor $\bar{B}$ contains this column. This means that its corresponding column in F will also be a unit vector, and so not make any contribution to G. By augmenting $\begin{bmatrix} G \\ H \end{bmatrix}$ with this unit vector we can obtain an enlarged G' with

$$G' = \begin{pmatrix} G & 0 \\ h & 1 \end{pmatrix}$$

where h is the corresponding row of H. Then the last row of $(G')^{-1}$ will contain all the nonzeros of the ℓ^{th} row of F^{-1} and will moreover

be precisely the μ required in the third equation of (14). Indeed if we partition $c = e_\ell$ into (c_1, c_2), then $c_2 = 0$ and $c_1 = (0,\ldots,1)$.

Thus in (14), $\lambda = 0$ during phase II. We can therefore compute the simplex multipliers, π, by solving two systems of equations

$$(G')^T \mu = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

and (16)

$$\bar{B}^T \pi = \begin{bmatrix} \mu \\ 0 \end{bmatrix}$$

and so achieve a substantial savings over the three systems in (14).

We conclude this section by tabulating the methods used to solve the systems (4) and (5), and also the work involved in terms of the number of smaller systems to be solved.

Table 1

System	Method of Solution	# small systems	
		$\bar{B}$	G
$By = c$	$\bar{B}z = a$ $G y^1 = z^1$ $\bar{B}w = a - B^1 y^1$ where $w = \begin{bmatrix} 0 \\ y^2 \end{bmatrix}$	$[\frac{K}{2}] + K^*$	1
$B^T \pi = c$ Phase I	$\bar{B}^T \lambda = \begin{bmatrix} 0 \\ c^2 \end{bmatrix}$ $G^T \mu = c^1 - (B^1)^T \lambda$ $\bar{B}^T \pi = \begin{bmatrix} \mu \\ c^2 \end{bmatrix}$	2K	1
$B^T \pi = c$ Phase II	$G^T \mu = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$ $\bar{B}^T \pi = \begin{bmatrix} \mu \\ 0 \end{bmatrix}$	K	1

*On the average.

6. Updating the Factorization

Suppose that we begin an iteration of the simplex method with a basis B and a minimal factorization $\bar{B}F$ of B. At the end of this iteration we obtain a new basis B' by replacing a column b_i of B with a column a of A. We then need to obtain a minimal factorization $\bar{B}'F'$ of B' with as little work as possible.

While we only use the submatrix G of F, it will be convenient here to first determine how to update f and then induce the update on G.

The two major steps of the update will be as follows:

(i) try to bring the column a into $\bar{B}$ without disturbing square block triangularity;

(ii) restore the factorization to minimality by seeing if there are any other columns of B, not currently in $\bar{B}$, that can be brought into $\bar{B}$.

As we shall show, the update on $\bar{B}$ can be accomplished by updating at most two of the factorizations of its diagonal blocks. These updates take the form of an exchange of columns. The update on G will consist of a possible change in size, up or down by one, and the addition of at most two rank 1 matrices, each of the form uv^T where u is a column vector and v^T is a row vector.

6.1. Review of Updating an Inverse Representation of a Matrix When an Exchange of Columns has Taken Place

Consider a square nonsingular matrix M, and suppose that we wish to modify M by replacing one of its columns with some given column d. Suppose further that the columns we allow to be replaced by d are restricted to some submatrix N of M.

Compute the representation y of d by solving

$$My = d \quad .$$

If an exchange of columns is at all possible there will be a non-zero element y_i in y corresponding to some column m_i of M that is in the restricted submatrix N. Usually, by 'non-zero' we would mean that

$$|y_i| / \|y\|_\infty \geq TOL$$

where TOL is a preassigned tolerance, typically of the order 10^{-4}.

The modified M may then be written as

$$M' = M + (d - m_i)e_i^T \quad ,$$

i.e.,

$$M' = M[I + (y - e_i)e_i^T] \quad .$$

The inverse of the elementary matrix $I + (y - e_i)e_i^T$ is another elementary matrix $I - \frac{1}{y_i}(y - e_i)e_i^T$ so that the modified inverse of M may be written as

$$(M')^{-1} = [I - \frac{1}{y_i}(y - e_i)e_i^T]M^{-1} \quad .$$

When the update is actually implemented in this form, it is called 'updating in product form' (see e.g., Dantzig [6]). However, there are today several other methods available to accomplish this update. These depend on the representation of M^{-1} as well as the degree to which one wishes to preserve sparsity and/or numerical stability. See for example Forrest and Tomlin [9], and Saunders [23].

6.2. Updating $\bar{B}$

Let the entering column a replace the i^{th} column b_i of B. Let a be in period t i.e., a has the form

$$a = \begin{bmatrix} 0 \\ a_t \\ \vdots \\ a_K \end{bmatrix}$$

We wish to determine whether a can replace a column of $\bar{B}_t$ where

$$\bar{B}_t \triangleq \begin{bmatrix} 0 \\ \bar{B}_{tt} \\ \vdots \\ \bar{B}_{Kt} \end{bmatrix} .$$

From the calculation of the representation of a in terms of B (see equations (13)) we already have z_t, the representation of a_t in terms of $\bar{B}_{tt}$. Thus a_t can replace any column of $\bar{B}_{tt}$ having a nonzero coefficient in z_t. At this point the following are possible.

6.2.1.

(a) The column b_i leaving B is in $\bar{B}_t$, and has a nonzero coefficient in z_t. Column a then replaces b_i in $\bar{B}$ by an exchange of their t^{th} period components in $\bar{B}_{tt}$. (b) Notice that we have changed $\bar{B}_{tt} \cap B_{tt}$, the basis for B_{tt}, and it is now possible that $\bar{B}'_{tt} \cap B'_{tt}$ is no longer a basis for B'_{tt}. Thus to restore the factorization to minimality we scan the columns of B'_{tt} that are not in $\bar{B}'_{tt}$ for a candidate to replace some column of $\bar{B}'_{tt}$ that is not in B'_{tt}. This can be done as indicated in Section 4.4.

If 6.2.1 is not the case then:

6.2.2.

Assuming that the column a is not already in $\bar{B}$, we try to introduce it into $\bar{B}_t$ in place of a column that is not in B. If there is no suitable nonzero in z_t to permit this, the column a cannot enter $\bar{B}$.

6.2.3.

The column b_i leaving B may be in $\bar{B}$. Suppose it is in period s (it is possible that $s = t$). We then scan the columns of B in period s that are not in $\bar{B}$ for a candidate to replace b_i in $\bar{B}_s$. This can be done by generating the corresponding row of $\bar{B}_{ss}^{-1} B_{ss}$ and then picking the column corresponding to a suitable nonzero in this row. Thus if b_i is the k^{th} column of $\bar{B}_s$, we solve

$$\bar{B}_{ss}^T w = e_k$$

and the required row is then $v^T = w^T B_{ss}$. If we succeed in finding a suitable nonzero in v^T corresponding to column b_j, say of B, we then represent b_j in terms of $\bar{B}$ by solving $\bar{B} r = b_j$.

From 6.2.1, 6.2.2 and 6.2.3 we see that the update on $\bar{B}$ thus takes the form of an exchange of columns in either $\bar{B}_{tt}$ or $\bar{B}_{ss}$ or both, where t and s respectively are the periods to which the entering and leaving columns belong.

6.3. Updating G

In updating G we must consider the effect on F of the column interchanges in B and in $\bar{B}$. It is best to treat the cases in 6.2 jointly with several subcases. Combinations of the following are possible:

I: a is column $\bar{b}_\ell$ in $\bar{B}$,

I^*: a is not a column of $\bar{B}$,

II: b_i is column $\bar{b}_k$ of $\bar{B}$,

II^*: b_i is not a column of $\bar{B}$,

III: a replaces b_i $(= \bar{b}_k)$ in $\bar{B}$,

III^*: a replaces $\bar{b}_\ell$ in $\bar{B}$ where $\bar{b}_\ell$ is not a column of B,

III^{**}: a does not enter $\bar{B}$,

IV: b_i $(= \bar{b}_k)$ is replaced in $\bar{B}$ by column b_j of B, where b_j is not a column of $\bar{B}$,

IV^*: column b_i $(= \bar{b}_k)$ is not replaced in $\bar{B}$,

V: $\bar{b}_\ell$, not in B, is replaced by b_j, not in $\bar{B}$.

These can occur as follows: (I^*, II, III), (I^*, II, III, V), (I^*, II, III^*, IV^*), (I^*, II, III^*, IV), $(I^*, II, III^{**}, IV^*)$, (I^*, II, III^{**}, IV), (I^*, II^*, III^{**}), (I^*, II^*, III^*), (I, II^*), (I, II, IV^*), (I, II, IV).

We shall treat them individually.

I^*, II, III: a replaces the i^{th} column of B, b_i, $b_i = \bar{b}_k$. Note that the i^{th} column of F is unit vector e_k. Using the approach of section 6.1 we may write

$$B' = B[I + (y - e_i)e_i^T]$$

and

$$\bar{B}' = \bar{B}[I + (z - e_k)e_k^T]$$

where the vectors y and z are the representations of a in terms of B and $\bar{B}$ respectively. Since $B = \bar{B}F$ and $B' = \bar{B}'F'$ it follows that

$$F' = [I - \frac{1}{z_k}(z - e_k)e_k^T] \; F \; [I + (y - e_i)e_i^T] \; .$$

Multiplying F into the third factor and noting that $Fy = z$ and $Fe_i = e_k$ we get

$$F' = [I - \frac{1}{z_k}(z - e_k)e_k^T] \; [F + (z - e_k)e_i^T] \quad .$$

This may be conveniently written as

$$F' = F - \frac{1}{z_k} z \, e_k^T F - e_k(e_i^T - e_k^T F) + \frac{1}{z_k} z \, e_i^T \quad .$$

We now determine the update on G by examining the corresponding rows and columns of the terms on the right hand side of this relation. Firstly, note that since the entering column a has replaced the same column in both B and $\bar{B}$, the columns and rows used by G' are the same as those used by G. Secondly, since row k and column

i of F are not used by G, the unit vectors e_k and e_i^T have only zeros in the rows and columns used by G. Therefore the last two terms play no part in the update of G, and we may write

$$G' = G + uv^T$$

where u and v^T are the corresponding subvectors of $-\frac{1}{z_k} z$ and $e_k^T F$, respectively.

Since z is already available to us from an intermediate step in the solution of By = a, we have u immediately at hand. However, $e_k^T F$, the k^{th} row of F, has no components in G and needs to be computed from scratch. To do this we solve

$$\bar{B}^T w = e_k$$

for w, the k^{th} row of $\bar{B}^{-1}$, and then set

$$v^T = w^T B^1$$

where B^1 consists of those columns of B that are not in $\bar{B}$.

I*, II, III, V: Having performed the first update in (I*, II, III) we have found column b_j of B' to replace column $\bar{b}_\ell$ of $\bar{B}'$ where both b_j and $\bar{b}_\ell$ are in period t and are not common to B' and $\bar{B}'$. Let r satisfy

$$\bar{B}' r = b_j \ .$$

Then we obtain

$$F'' = [I - \frac{1}{r_\ell}(r - e_\ell)e_\ell^T]F'$$

i.e.,

$$F'' = F' - \frac{1}{r_\ell}(r - e_\ell)e_\ell^T F'$$

Since the introduction of b_j into $\bar{B}'$ increases the number of common columns, the size of G' will decrease by 1. Partition G' as

$$G' = \begin{bmatrix} G'_{11} & g'_{12} \\ g'_{21} & g'_{22} \end{bmatrix}$$

where

$$\begin{bmatrix} g'_{21} \\ g'_{22} \end{bmatrix}$$

is the appropriate subvector of r and $[g'_{12} \quad g'_{22}]$ is the appropriate subvector of $e_\ell^T F'$ [1]. Then

$$G'' = G'_{11} - \frac{1}{r_\ell} g'_{21} g'^T_{12}$$

[1] Here $g'_{22} = r_\ell$.

I*, II*, III**: In this case

$$\bar{B}' = \bar{B}$$

$$B' = B[I + (y - e_i)e_i^T]$$

so that

$$F' = F[I + (y - e_i)e_i^T]$$

i.e.,

$$F' = F + (z - f_i)e_i^T$$

where f_i is the i^{th} column of F. Since b_i is not a column of $\bar{B}$, f_i is not a unit column and the above statement simply tells us to replace column i of F with z. The same applies to G, i.e.,

$$G' = G + uv^T$$

where u and v are the appropriate subvectors of $z - f_i$ and e_i^T respectively. Note that the portion of f_i corresponding to G is a column of G and can thus be found from the representation of G.

I*, II*, III*: Proceeding as in before we see that

$$F' = [I - \frac{1}{z_\ell}(z - e_\ell)e_\ell^T] [F + (z - f_i)e_i^T] \ .$$

This may be written as

$$F' = F - \frac{1}{z_\ell}(z - e_\ell)(e_\ell^T F - f_{\ell i}\, e_i^T) + (e_\ell - f_i)e_i^T$$

where f_i is the i^{th} column of F and $f_{\ell i}$ is the $(\ell,i)^{th}$ element of F. Since we have added column a to the common columns of B and $\bar{B}$, the size of G will be reduced by 1. This is born out by the term $(e_\ell - f_i)e_i^T$ in the above relation. It says that f_i is to be replaced by unit vector e_ℓ. Note also that the second term in the above right hand side makes no contribution to the i^{th} column of F' since the i^{th} component of $e_\ell^T F - f_{\ell i}\, e_i^T$ is zero.

Partition G as

$$G = \begin{bmatrix} G_{11} & g_{21} \\ g_{21}^T & g_{22} \end{bmatrix}$$

where

$$\begin{bmatrix} g_{21} \\ g_{22} \end{bmatrix}$$

is the subvector of f_i corresponding to G, and $[g_{21}^T \quad g_{22}]$ is the subvector of $e_\ell^T F$ corresponding to G.[1] Then

[1] Note that $g_{22} = f_{\ell i}$.

$$G' = G_{11} + u\, g_{21}^T$$

where u is the subvector of $-\frac{1}{z_k} z$ corresponding to G_{11}. g_{21}^T can be easily found from the representation of G.

I*, II, III**, IV*: This is similar to the case (I*, II*, III**) except that f_i is now unit vector e_k. Thus

$$F' = F + (z - e_k)e_i^T ,$$

and the size of G will increase by 1. Letting g denote the subvector of z corresponding to G, and h^T the subvector of the k^{th} row of F corresponding to G, we get

$$G' = \begin{bmatrix} G & g \\ h^T & z_k \end{bmatrix} .$$

To obtain h^T we proceed as in (I*, II, III).

I*, II, III**, IV: Following 6.2.3 suppose that r satisfies $\bar{B}\, r = b_j$. Then an analysis similar to that of 6.3.1 yields

$$F' = [\, I - \frac{1}{r_k}(r - e_k)e_k^T] \; [F + (z - e_k)e_i^T] \quad .$$

We can best view this as a two step update. Let

$$F^* = F + (z - e_k)e_i^T \quad .$$

Then

$$F' = F^* - \frac{1}{r_k}(r - e_k)e_k^T F^*$$

F^* is simply F with its i^{th} column, e_k replaced by z, and reflects column a replacing b_i in B. Since r is the representation of b_j in terms of $\bar{B}$, r is the j^{th} column of both F and F^*. Thus the transition from F^* to F reflects b_j replacing b_i $(= \bar{b}_k)$ in $\bar{B}$. Note that the j^{th} element of $e_k^T F^*$ is r_k, so that the j^{th} column of F' becomes e_k.

Partition G as $[G_1 \quad g]$ where g is the corresponding subvector of r. Let u denote the subvector of z corresponding to G, and let k^T denote the subvector of $e_k^T F$ corresponding to G_1. Then

$$G' = [G_1 \quad u] - \frac{1}{r_k} g[h^T, z_k] \quad .$$

From the update on $\bar{B}$ we have both r and z so that g, u, r_k, and z_k are immediately available. However, h^T needs to be computed as in (I^*, II, III).

<u>I^*, II, III^*, IV^*</u>: Here we obtain

$$F' = [I - \frac{1}{z_\ell}(z - e_\ell)e_\ell^T] \; [F + (z - e_k)e_i^T]$$

which simplifies to

$$F' = F + (e_\ell - e_k)e_i^T - \frac{1}{z_\ell}(z - e_\ell)e_\ell^T F \quad .$$

The term $(e_\ell - e_k)e_i^T$ reflects the fact that after the update, column ℓ of $\bar{B}$ is a common column and column k is not, whereas before the update, the reverse was the case. Let h^T denote the row of G corresponding to the k^{th} row of F, and let v^T denote the subvector of the ℓ^{th} row of F corresponding to G. Then the above statement means that we need to exchange v^T and h^T in G before continuing with the rank one modification. Thus if we partition G as

$$G = \begin{bmatrix} G_1 \\ h^T \end{bmatrix} \quad ,$$

we get

$$G' = \begin{bmatrix} G_1 \\ v^T \end{bmatrix} + uv^T$$

where u is the subvector of $-\frac{1}{z_\ell} z$ corresponding to the rows of

$$\begin{bmatrix} G_1 \\ v^T \end{bmatrix} \quad .$$

We compute v^T as in (I^*, II, III) and obtain h^T from the representation of G.

I^*, II, III^*, IV: We can think of this case as a continuation of the previous case, viz. perform the update on G as described above, and then introduce b_j into $\bar{B}'$ in place of b_i ($= \bar{b}_k$). This can be accomplished in the same way as the case (I^*, II, III, V) with ℓ replaced by k.

I, II, IV^*: This is the same as the case (I^*, II, III^*, IV^*) with z specialized to e_ℓ, and the resulting u therefore being zero.

I, II, IV: Proceed exactly as in (I^*, II, III^*, IV).

I, II^*: We may regard this as a special case of (I^*, II^*, III^*) with z replaced by e_ℓ and the resulting u being zero.

6.4. Discussion and Summary of the Fundamental Updating Operations

In Sections 6.2 and 6.3 we enumerated the (many) possible updates that can occur. That these are the only possibilities and that minimality is indeed preserved at each iteration has been claimed but perhaps not established with full rigor. However these statements can be easily verified with the use of Theorem 4.3.

Presenting the update -- and for that matter, implementing it -- has been a laborious task. However, it is very much this aspect of an implementation of the simplex method that determines its performance relative to that of other implementations. The updating procedure usually requires a significant time slice, and determines the stability

and speed of the remaining iterations. Reliable estimates of these effects can only be obtained by an in depth analysis.

A glance at the updates of the previous sections shows that there are several operations that many of them have in common. These are:

(i) INVUPD -- updating the factorization of a $\bar{B}_{tt}$,

(ii) FROW -- solving the system of equations $\bar{B}^T w = e_k$, and then computing $w^T B^1$ to yield the structural part of the k^{th} row of F,

(iii) FCOL -- solving the system of equations $\bar{B} r = b$ to obtain the representation r of some column b in terms of $\bar{B}$,

(iv) GROW -- finding a row of G,

(v) GCOL -- finding a column of G,

(vi) RSWAP -- replacing a row of G,

(vii) CSWAP -- replacing a column of G,

(viii) ADD -- increasing the dimension of G by 1,

(ix) DEL -- decreasing the dimension of G by 1,

(x) RANK1 -- performing a general rank one update on G, that is $G' = G + uv^T$ where u and v are arbitrary vectors,

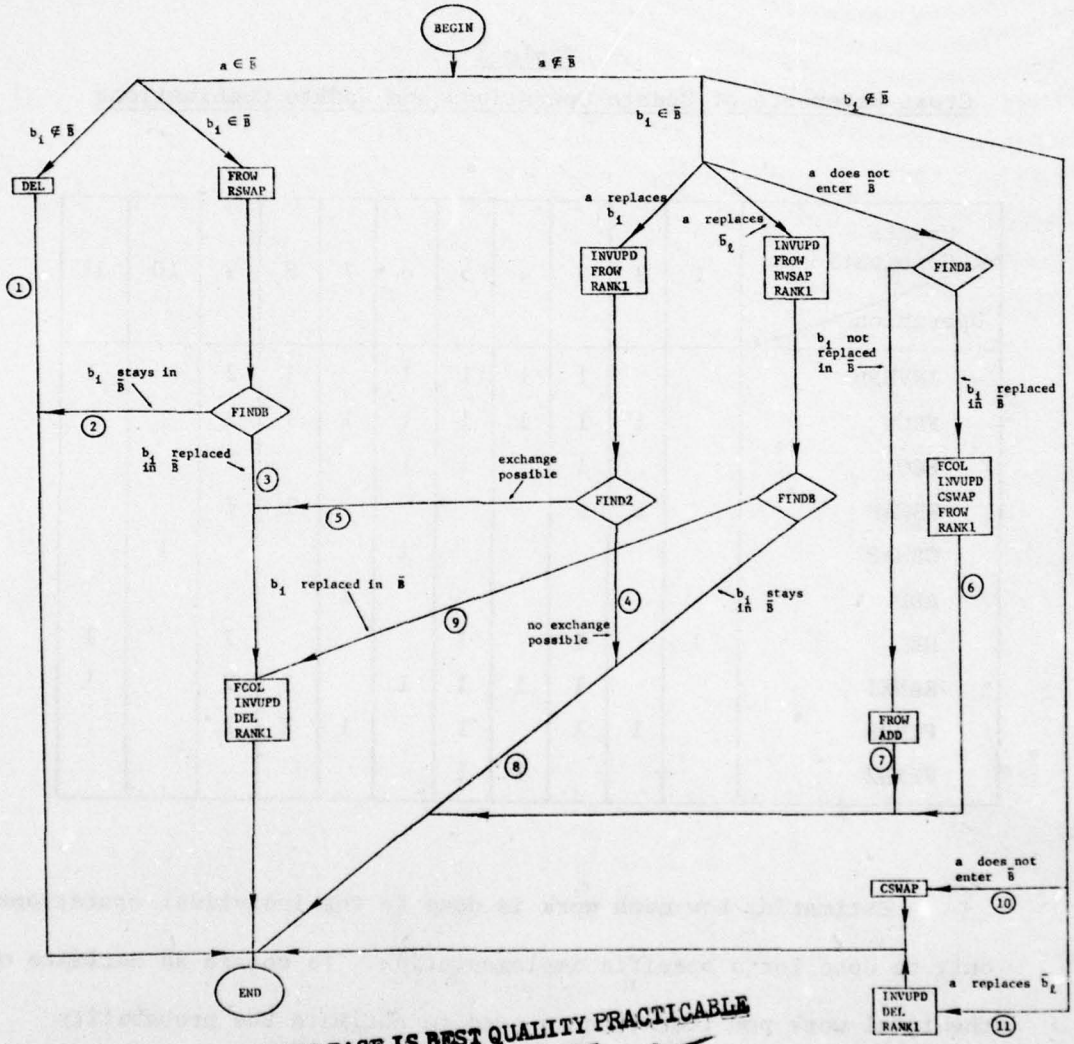
(xi) FINDB -- seeks a column of B that is not in $\bar{B}$ to replace column b_i in $\bar{B}$,

(xii) FIND2 -- seeks two columns, one in B that is not in $\bar{B}$ to replace one in $\bar{B}$ that is not in B.

We may now use these operations to depict the whole updating process in the accompanying flow diagram in Figure 1.

Note that the GCOL and GROW operations do not appear in the flow diagram since they are used implicitly in the ADD and DEL operations.

Figure 1
The Update Flow Diagram



From the flow diagram, we obtain the following cross reference table for the 11 update combinations (indicated by the circled numbers) and 10 operations. The entries in the table are the number of times an operation is used in an update combination.

Table 2

Cross Reference of Update Operations and Update Combinations

Update Combination / Operation	1	2	3	4	5	6	7	8	9	10	11
INVUPD			1	1	1	1		1	2		1
FROW		1	1	1	1	1	1	1	1		
FCOL			1		1	1			1		
RSWAP		1	1					1	1		
CSWAP						1				1	
ADD							1				
DEL	1		1		1				1		1
RANK1			1	1	1	1		1	2		1
FINDB		1	1		1		1	1	1		
FIND2				1	1						

Estimating how much work is done in the individual operations can only be done for a specific implementation. To obtain an estimate of the total work per iteration we need to estimate the probability distribution of the 11 possible update combinations. While this is dependent on the specific problem being solved, a reasonable degree of uniformity has been observed.

In the sections following we shall address these questions by describing a specific implementation and by analyzing the statistics of a number of test runs.

7. Implementation

The foregoing details of the factorization were all implemented in an in-core Fortran program LPBLK on the IBM 370/168 at the Stanford Linear Accelerator Center. Suitable existing software was adapted wherever possible, with the remainder of the program being written in Mortran [4][1]. Mortran was chosen for two reasons: Firstly, it allows one to do structured programming, a key ingredient of any developmental work in this area. Secondly, it retains all the important features of Fortran: fast run times, ability to use the Watfiv compiler, and comparability with other implementations, few of which are written in any language other than Fortran.

Data is input in MPS/360 format [18] including simple upper and lower bounds on any of the variables. The code also has the capability of starting the simplex algorithm from any prespecified basis, input either in MPS/360 format or in the form of a bitmap, which is a vector of n elements, each indicating whether a variable is basic or nonbasic at its upper or lower bound. Such a capability allows one to make detailed comparisons with other codes, especially on large problems where running the problem from cold start to optimality is both less instructive and of prohibitively large cost.

---

[1] Mortran was developed at the Stanford Linear Accelerator Center by J. Cook and L. Shustek. It is a structured language together with a Macro processor to translate the language into Fortran. Any Fortran statement is a Mortran statement.

Several of the important routines in LPBLK are adpatations of routines written by John Tomlin. Indeed, LPBLK is modeled on his experimental code LPM1 [28], which is an implementation of the revised simplex algorithm using an LU decomposition of the basis with standard product form update.

As a standard for comparison, we chose another in-core Fortran code, MINOS [20] developed by B. Murtagh and M. Saunders. The linear programming part of MINOS is a fast and stable implementation of the revised simplex method, taking no advantage of any structure other than sparsity. It was easy to implement in LPBLK the CHUZR (choose pivot row) and PRICE (choose entering column) routines of MINOS so that any differences in storage and run time could be accounted for by the different methods of representing the basis inverse. For the main basis factorization and updating details of MINOS, see [23]. The routines of MINOS are all described in [24].

7.1. Factorizing and Updating the $\bar{B}_{tt}$

Since the $\bar{B}_{tt}$ are sparse and are updated by an exchange of columns, it was natural to treat them as one would the whole basis of a general sparse linear program. Accordingly we chose to do an LU factorization of $\bar{B}_{tt}$ and to update it using the method of Forrest and Tomlin [9].

For a given $\bar{B}_{tt}$, the LU factorization is computed as follows. Permute $\bar{B}_{tt}$ to the form

$$\begin{bmatrix} L_1 & & \\ C & D & \\ E & F & L_2 \end{bmatrix}$$

where L_1 and L_2 are lower triangular and D is the smallest such matrix. D is called the "bump." In order to prevent fill-in occurring in F, reorder the rows and columns to obtain

$$\begin{bmatrix} L_1 & & \\ \hat{E} & U_1 & \tilde{F} \\ C & & D \end{bmatrix}$$

where U_1 is L_2 permuted to form an upper triangular matrix. The problem is thus reduced to finding an LU decomposition of D. This can be done in a sparse manner by preordering the columns of D in ascending order of "merit," i.e., compute

$$M_j = \sum_{d_{ij} \neq 0} (r_i - 1)$$

where r_i is the number of non-zeros in row i, and reorder the columns of D so that

$$M_1 \leq M_2 \leq \cdots$$

Each M_j is an estimate of the potential number of nonzeros that can be created by pivoting somewhere in column j of D. Further details of this method may be found in [25].

Forrest and Tomlin updating is aimed at preserving the sparsity of the LU factors. Although its stability properties are not ideal, there is a convenient accuracy test (see [26]) that can be used to decide whether or not reinversion is necessary.

7.2. Factorizing and Updating G

From Section 6, we see that whichever factorization we choose for G, it will have to be such that rank one updates and changes in size can be easily and stably accomplished. Unlike the $\bar{B}_{tt}$, G is a computed matrix so that the need for stable methods is even greater for G than for the $\bar{B}_{tt}$.

In the light of these considerations, a QR factorization of G seemed to be the logical choice. Since G turns out to be typically between 30% and 50% dense there would be little need for sparsity considerations, thus allowing the implementation to be relatively straight forward.

For a detailed analysis of the QR factorization and methods of modifying the Q and R factors, see [15]. We shall provide only an outline of the main procedures.

The factors Q and R are defined to be those orthogonal and upper triangular matrices respectively that satisfy

$$Q\,G = R \quad .$$

If G is nonsingular then Q and R are unique up to some signs. One method of computing these factors is to apply a sequence of elementary Householder transformations to G to reduce it to the matrix R. The j^{th} Householder transformation is an orthogonal matrix of the form

$$P_j = I - 2\, u_j\, u_j^T$$

where u_j is a vector satisfying $u_j^T u_j = 1$. After r of these transformations have been applied we obtain

$$P_r \cdots P_2 P_1 G = \begin{bmatrix} x & x & x & x & x & x \\ & x & x & x & x & x \\ & & x & x & x & x \\ & & & x & x & x \\ & & & x & x & x \\ & & & x & x & x \end{bmatrix} \begin{matrix} \\ \\ \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} r \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} p - r \end{matrix}$$

where G is $p \times p$ and the remaining $(p - r) \times (p - r)$ bottom right hand matrix has yet to be reduced to upper-triangular form. Thus the vectors u_j have $p - j + 1$ nonzero components. After $p - 1$ applications we obtain the desired R on the right hand side and may form Q by evaluating the product

$$Q = P_{p-1} \cdots P_2 P_1$$

from left to right.

The routines to perform this factorization were written by Margaret Wright [30]. One of the features of her routines is that one has the option to specify one of several column interchange strategies, e.g., to choose as the next column to be reduced, the column of largest Euclidean length in the remaining matrix. Having several interchange strategies at one's disposal is important when dealing with ill-conditioned systems. Further, using the abovementioned strategy ensures that the diagonal elements of R are in descending order of magnitude and in addition that

$$|r_{ii}| \geq |r_{jk}| \qquad j,k \geq i \ .$$

This enables one to use the ratio

$$\rho = |r_{11}|/|r_{pp}|$$

as a lower bound on the condition number $\kappa(G)$ (see [15]). The condition number of G is such that if perturbations of order ε are made in the data (G, g) of the system

$$G x = g \ ,$$

the exact solution x will be perturbed by at most $K(G)\,\varepsilon$. Thus excessively large values of ρ may be taken as an indication that G

is ill-conditioned and that it or the whole factorization should be recomputed perhaps with a different choice of $\bar{B}$. (See Section 9.3.)

When G is modified by a matrix of rank one, the factors Q and R may be updated by applying a sequence of plane rotations. These are orthogonal matrices of the form

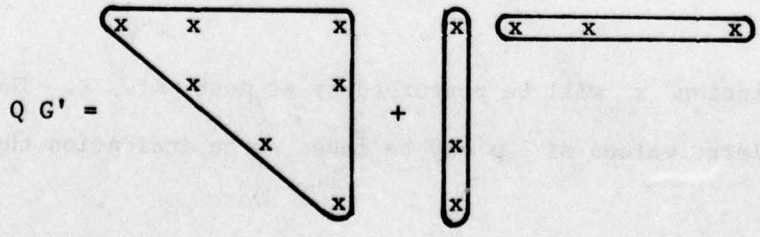
$$T = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & & s & & \leftarrow i^{th} \\ & & & \ddots & & & \\ & & s & & -c & & \leftarrow j^{th} \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}$$

where for a given vector x the numbers c and s are chosen so that T applied to x annihilates one of x_i, x_j and leaves elements x_k, $k \neq i,j$ unaltered.

Suppose that $G' = G + u\, v^T$ and $Q\, G = R$. Then

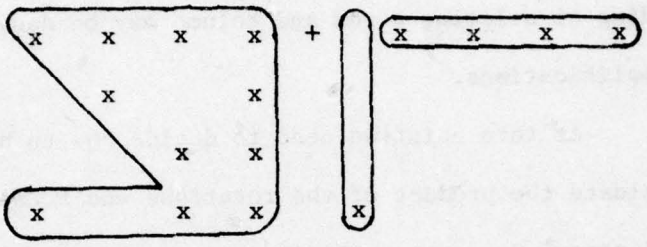
$$Q\, G' = R + \bar{u}\, v^T$$

where $\bar{u} = Q\, u$. Diagramatically we have



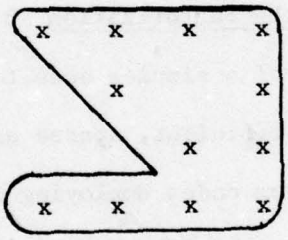
We now apply rotations in the planes $(p-1, p)$, $(p-2, p)$, ..., $(1, p)$ respectively to annihilate the elements $\bar{u}_{p-1}, \ldots, \bar{u}_1$ in this order. This yields

$$T_{p-1} \cdots T_1 \, Q \, G' =$$



This operation is called a backward sweep. Adding the elements of the second term on the right to the bottom row of R yields

$$T_{p-1} \cdots T_1 \, Q \, G' =$$



A forward sweep is now performed to annihilate the last row of R to restore it to upper-triangularity. This is accomplished by means of rotations in the planes $(1, p)$, $(2, p)$, ..., $(p-1, p)$ respectively. We obtain

$$T_{2p-2} \cdots T_p \, T_{p-1} \cdots T_1 \, Q \, G' = R'$$

so that

$$Q' = T_{2p-2} \cdots T_p \, T_{p-1} \cdots T_1 \, Q \quad .$$

Note that the order in which the rotations are applied is crucial. Adding or deleting a row and column may be done similarly, with certain simplifications.

At this point we need to decide how to handle Q. We can either evaluate the product of the rotations and Householder transformations and store Q as an explicit dense square matrix, or maintain Q in product form. There are significant drawbacks and advantages associated with both methods, and we shall analyze them in detail in Section 8.

7.3. Recomputing the Factorization $B = \bar{B}F$

The ability of a simplex code to recompute the representation of the basis in an efficient, sparse and stable manner is often what distinguishes it from codes employing different basis representations. In a typical run, one would refactorize the basis say every 50-100 iterations in order to reduce the number of nonzeros currently in the representation and also to retain numerical accuracy. When starting the run from a prespecified basis, the initial step is to refactorize this basis before proceeding.

In Section 4 we sketched a method of finding a minimal factorization for any given B. This involved finding a maximal linearly independent subset of columns of each B_{tt} and then augmenting them

appropriately with unit columns to obtain the corresponding $\bar{B}_{tt}$. It turns out that the method of computing the LU factors of a sparse non-singular matrix outlined in Section 7.1 can be easily adapted for this task.

We begin by permuting B_{tt} to the form

$$\begin{bmatrix} L_1 & & \\ C & D & \\ E & F & L_2 \end{bmatrix}$$

as before, the only difference being that D is now rectangular. D may have more or fewer rows than columns, and may be of any rank. Proceeding exactly as before we compute the "merit" counts of each column of D and rearrange them so as to be in ascending order of merit.

We now simply pivot in this order, skipping a column when there is no nonzero of some suitable magnitude available as a pivotal element. After one such pass, a second pass can be made through the set of rejected columns in case some of them are now suitable for pivoting. Finally, we insert unit columns in those rows in which no pivoting has yet taken place.

The "merit" counts used in this manner are still reasonably good estimates of the potential number of nonzeros that can be created

by pivoting in a column. This is because like the matrices B_{tt}, D is usually not very far from being square, and is close to being of full rank.

It is usually difficult to answer the question, what is the rank of a matrix, or more precisely in our context, what is our threshold for determining whether or not the next column is linearly dependent on the current set. Observe that in our case, stopping a few columns short of obtaining a maximal independent set is of little consequence since it only implies that we shall be working with a factorization that is slightly off from being minimal. Thus in the interests of stability it would pay us to use stricter tolerances than in the case when we know what the rank of a matrix should be.

Once we have found our factor $\bar{B}$ we represent, in terms of $\bar{B}$, each column of B that was rejected in the above process. Thus if p columns were rejected, we solve p sets of equations of the type

$$\bar{B}y = b_i \qquad \text{(some } i\text{)}$$

and form G by extracting from each y those components corresponding to the augmented unit vectors.

7.4. Looking at the Fundamental Operations

We are now in a position to reconsider the factorization algorithm with respect to the fundamental operations outlined in Section 6.4. From the above implementation, it can be seen that each of these operations

can be broken down into combinations of the following basic operations:

TRANB -- a transformation involving one of the $\bar{B}_{tt}$,

TRANQ -- a transformation involving Q,

TRANR -- a transformation involving R,

SWEEP -- applying a sequence of rotations to annihilate a vector of nonzeros.

By considering the specific details of the implementation we arrive at the following table.

Table 3

Cross Reference of Operations in the Update

	TRANB	TRANQ (Q EXPLICIT)	TRANQ (Q PRODUCT)	SWEEP
INVUPD	.5			
FROW	$[\frac{K}{2}]$			
FCOL	$[\frac{K}{2}]$			
RSWAP		1.5	2	.5
CSWAP			1	2
ADD		1	1	2
DEL		.5	2	1.5
RANK1		1	1	2
FINDB	1			
FIND2	$\frac{P}{K}$			

Each entry is the average number of times that the operation heading its column is used by the operation heading its row. For example, the RSWAP operation uses 1.5 TRANQ operations when Q is stored explicitly.

Note that the TRANR operation is not used during the update. It is used only twice per iteration, in the solution of equations (4) and (5). Also from Table 1 we see that the TRANB operation is in addition used $3K + [\frac{K}{2}]$ times during Phase I and $2K + [\frac{K}{2}]$ times during Phase II.

8. Estimating the Work per Simplex Iteration Accounted for by the Factorization

Obtaining a good estimate of how much work is involved in any implementation is difficult if not impossible. Factors such as the machine and compiler in question, array accesses, the characteristics of the problem being run and the like, all have an important bearing on the ultimate runtime. However, a criterion that is often used by numerical analysts is the total number of multiplications. While it is far from adequate, it is usually instructive to analyze an algorithm from this point of view.

We shall now do this by estimating how many multiplications per simplex iteration are accounted for by each of the two factors $\bar{B}$ and G. Closely related to estimating this quantity for G is the question of G in product form versus G stored explicitly.

8.1. Q Stored in Product form Versus Q Stored Explicitly

A priori we see that storing Q in product form saves us evaluating the product of the householder matrices as well as applying each rotation explicitly to Q. However transformations involving Q become more expensive, and in cases where a row or column of Q is desired it now has to be computed by transforming an appropriate unit vector. Further, as the factors accumulate, the need may arise for a more frequent refactorization of G.

In the following we shall assume that the size of G remains roughly constant at $p \times p$ and that G and its QR factorization are recomputed every q iterations. We shall also require the following statistics

(i) α : the average number of sweeps per iteration,

(ii) ρ : the average proportion of rotations per sweep,

(iii) γ : the average number of transformations involving Q, i.e., TRANQ operations,

(iv) η : the average number of nonzeros in the representation of a $\bar{B}_{tt}$.

Note that in working with η, we are assuming that the $\bar{B}_{tt}$'s are all comparable in size and further that the frequency for recomputing the factorization $B = \bar{B}F$ has been fixed beforehand, so as to keep η constant.

Let us begin by estimating the number of multiplications required to generate the p columns of F of which G is a submatrix. Solving a system of the type

$$\bar{B}y = b_i$$

requires on the average the solution of $[\frac{K}{2}]$ [1] small systems involving the $\bar{B}_{tt}$.

---

[1] When the number of periods is small this is a low estimate since most of the columns determining G arise in the initial periods.

Solving systems of the type

$$\bar{B}_{tt}\, y_t = \text{rhs}$$

will require at most η multiplications. For sparse right hand sides one would usually require significantly fewer multiplications. However the right hand sides become progressively more dense as t increases. On the other hand $[\frac{K}{2}]$ underestimates the number of small systems to be solved, so that we may reasonably hope that the two estimates off-set each other and conclude that, on the average, the number of multiplications to generate G is

$$p\,\eta\,[\tfrac{K}{2}] \quad .$$

8.1.1. Q Explicit

It is straightforward to show that recomputing the QR factorization and forming the product of the Householder transformations both require $2/3\ p^3 + O(p^2)$ miltiplications. Thus if we recompute and refactorize G every q iterations, we obtain a contribution of

$$\frac{1}{q}\{\, p\,\eta\,[\tfrac{K}{2}] + \frac{4}{3}\, p^3 + O(p^2)\}$$

multiplications.

Applying the rotation

$$\begin{pmatrix} c & s \\ s & -c \end{pmatrix}$$

to a vector of 2 components requires 4 multiplications. Since Q is square and R is triangular we find that a full sweep of ρp rotations requires $2\rho\, p^2 + 0(p)$ multiplications for R and $4\rho p^2 + 0(p)$ multiplications for Q, yielding a contribution of

$$6\rho\, p^2 + 0(p)$$

multiplications per sweep.

Also, TRANQ and TRANR operations require p^2 and $p^2/2 + 0(p)$ multplications respectively.

Noting that the number of transformations per iteration is γ for Q and 2 for R (one for the simplex multipliers and one for the representation of the incoming column) and that we are assuming α sweeps per iteration, we obtain a total estimate of

$$\frac{1}{q}\{p\,\eta\,[\frac{K}{2}] + \frac{4}{3}\,p^3\} + (6\,\alpha\,\rho + \gamma + 1)p^2 + 0(p)$$

multiplications per iteration, assuming that $q = 0(p)$. This is the total number of multiplications per iteration that can be attributed to G when Q is stored explicitly.

8.1.2. Q In Product Form

Here, recomputing Q and R costs us $2/3\ p^3 + O(p)$ multiplication per iteration since we do not compute the product of the Householder transformations.

Each transformation involving Q requires the application to a vector of $p - 1$ Householder transformations of diminishing size, a total of $p^2 + O(p)$ multiplications. Further, if there are r rotations currently in the representation of Q, these will require 4r multiplications. Since each sweep produces ρp rotations, there will be $\alpha\rho pj$ rotations after j iterations. The average of this quantity over q iterations is $1/2\ \alpha\rho pq + O(p)$ rotations. Therefore each transformation involving Q requires, on the average,

$$2\alpha\rho pq + p^2 + O(p)$$

multiplications.

As in 8.1.1 observe that a sweep applied to R requires $2p^2 + O(p)$ multiplications, and that transformations involving R require $p^2/2 + O(p)$ multiplications.

Doing α sweeps, γ transformations with Q and 2 with R we finally obtain a total estimate of

$$\frac{1}{q}\{\ p\ \eta\ [\frac{K}{2}] + \frac{2}{3}\ p^3\} + 2\alpha\rho\gamma pq + (\gamma + 2\alpha + 1)p^2 + O(p)$$

multiplications.

8.2. The Contribution of $\bar{B}$

At the end of Section 7.4 we noted that we require $3K + [\frac{K}{2}]$ TRANB operations during Phase I and $2K + [\frac{K}{2}]$ TRANB operations during Phase II. In addition we need to consider the number of TRANB operations used during the update.

Let ξ denote the number of FROW or FCOL operations, and μ the number of TRANB operations in the remaining update (INVUPD, FINDB, FIND2). Then the total number of TRANB operations is

$$3K + (\xi + 1)\ [\tfrac{K}{2}] + \mu$$

during Phase I and

$$2K + (\xi + 1)\ [\tfrac{K}{2}] + \mu$$

during Phase II.

Multiplying these by η yields the required number of multiplications.

8.3. Summary

We summarize in the following table.

Table 4

Average Total Number of Multiplications Per Iteration

$\bar{B}$	PHASE I	$\{3K + (\xi + 1)\ [\frac{K}{2}] + \mu\}\eta$
	PHASE II	$\{2K + (\xi + 1)\ [\frac{K}{2}] + \mu\}\eta$
G	EXPLICIT	$\frac{1}{q}\{p\ \eta[\frac{K}{2}] + \frac{4}{3}\ p^3\} + (6\alpha\rho + \gamma + 1)p^2$
	PRODUCT	$\frac{1}{q}\{p\ \eta\ [\frac{K}{2}] + \frac{2}{3}\ p^3\} + 2\alpha\rho\gamma pq + (\gamma + 2\alpha + 1)p^2$

These estimates are difficult to interpret without knowing what the values of the constants are. In Section 9 we shall use the test data to gain more insight into this analysis.

9. Computational Results and a Further Analysis

As our test problems we used 5 dynamic models. Their statistics are summarized below.

Table 5

Problem Statistics

Name	Rows	Structural Columns	Periods	Non-zeros	% Density	Bounds
PILOT8	626	1376	8	6026	.7	YES
SC205	204	202	19	551	1.3	NO
SCRS8	491	1169	4	4029	.7	NO
SCSD8	398	2750	39	11334	1.0	NO
SCTAP1	301	480	10	3372	2.3	NO

PILOT8 is an 8 period SIGMA version of the PILOT energy model currently under study at Stanford University. It has a staricase structure together with a few nonzeros sprinkled in the lower block triangle. For a description of the model see [8].

The other models are staircase models supplied by James Ho. They are all derived from real models in economic planning (SC205), agricultural production scheduling (SCRS8), engineering design (SCSD8) and dynamic traffic control (SCTAP1). For more detailed descriptions of these models see [13].

The purposes of presenting the data are twofold: (i) to obtain a basic understanding of the behavior of this factorization method, and (ii) to establish its potential as a means of solving large dynamic time period models. In the former case we recorded observations relevant to the analyses of the preceding sections, as well as making runs with varying degrees of partitioning, e.g., combining every 3 periods to form new periods each 3 times as large but 1/3 as many in number. In addition is recorded a run of PILOT8.S, the PILOT8 model rescaled using the geometric mean.[1] The scaling was done by Mohammed Aganagic.

The runs themselves were made in several different ways. For PILOT8 and PILOT8.S we started from an advanced basis and stopped the run after 2 minutes CPU time. This yielded approximately 500 iterations. The same advanced basis was used for all these runs. For SCSD8, we started from an advanced basis and allowed the runs to terminate at optimality, again requiring about 500 iterations. The remaining models were all started from an identity basis and allowed to terminate at optimality. The iteration counts for these were 230 for SC205, 870 for SCRS8, and 400 for SCTAP1.

---

[1] For each row compute $(\max_j |a_{ij}| \min_j |a_{ij}|)^{1/2}$ where the min is taken over $a_{ij} \neq 0$. Divide the row by this quantity. Proceed similarly for the columns. See [27].

9.1. Analysis of the Estimated Number of Multiplications per Iteration

We began by assessing the probability distribution of the eleven update combinations discussed in Section 6.4. This was done by recording the update at each iteration and then computing the relative frequencies. These appear in Table 11 at the end of this section.

Following this we used Table 11 together with the cross reference Table 2 to obtain the average frequency per iteration with which each of the operations INVUPD, FROW, ... is used. These figures appear in Table 12. The values for ξ, μ, α, and γ in Table 13 were computed from Table 12 together with the cross reference Table 3. The remaining constants in Table 13 were taken directly from the individual runs.

A glance at Table 13 shows that the constants ξ, α, γ, and to some extent μ and ρ, are remarkably independent of the model, the number of periods (K) and the size of G (p). Indeed it seems reasonable to conclude that for any problem similar in nature to the ones tested, we may take

$$\xi \approx 1$$

$$\mu \approx 2$$

$$\alpha \approx 2.5$$

$$\gamma \approx 3.2 \quad \text{Q explicit}$$
$$3.7 \quad \text{Q product}$$

$$\rho \approx .6$$

If we do this and approximate [K/2] by K/2, we obtain from Table 4 the following estimates of the number of multiplications involved.

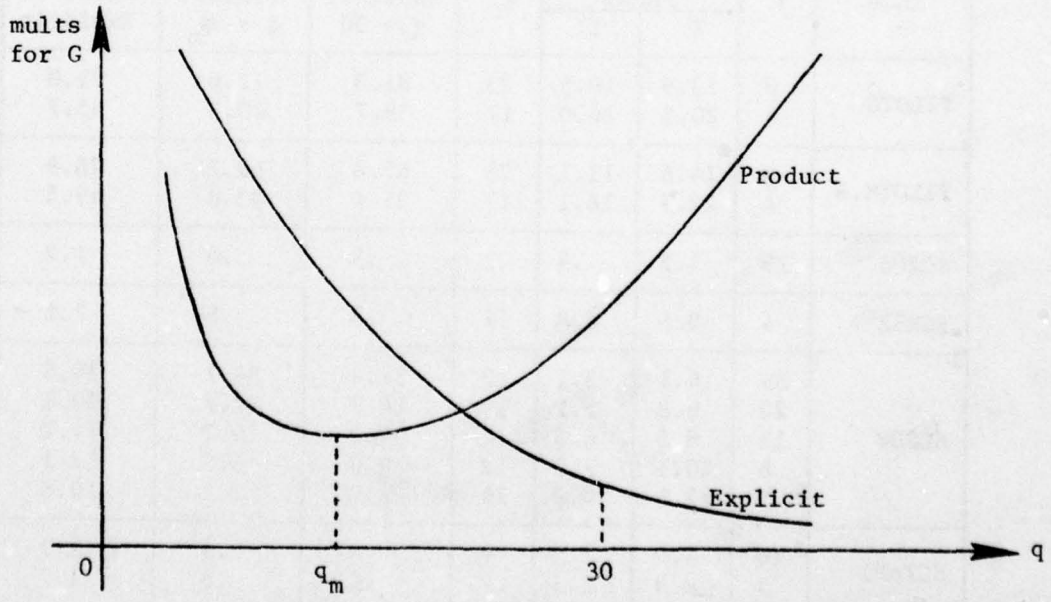
Table 6

Average Total Number of Multiplications per Iteration

$\bar{B}$	Phase I	$(4\ K + 2)\eta$
	Phase II	$(3\ K + 2)\eta$
G	Explicit	$\frac{1}{q}\{\frac{1}{2}\ p\ K\ \eta + \frac{4}{3}\ p^3\} + 13.2\ p^2$
	Product	$\frac{1}{q}\{\frac{1}{2}\ p\ K\ \eta + \frac{2}{3}\ p^3\} + 11.1\ p\ q + 9.7\ p^2$

The first question to consider is what is the best value for q, the frequency with which we recompute G and its factorization. Plotting the number of multiplications for G as a function of q yields the following graph.

Figure 2



With Q stored explicitly, it clearly never pays to recompute G when the number of multiplications is the only consideration. However, for stability reasons, it turns out to be essential to recompute G fairly frequently. A good rule of thumb seems to be $q \approx 30$.

With Q stored in product form, the theoretical minimizer, q_m (say), can easily be shown to be

$$q_m \approx .23\sqrt{K\,\eta + p^2} \quad .$$

The following table gives the number of multiplications for each of the test runs. These were found by substituting into Table 4 the observed values of the constants in Table 11.

Table 7

Total Multiplications/Iteration

Name	K	$\bar{B}$ (1000's) Phase I	$\bar{B}$ (1000's) Phase II	q_m	G (1000's) Explicit $q = 30$	G (1000's) Product $q = q_m$	Total Phase II Explicit
PILOT8	8	13.9	10.5	23	81.3	77.6	91.8
	4	20.5	16.0	17	39.7	40.3	55.7
PILOT8.S	8	14.6	11.1	23	67.8	69.2	78.9
	4	22.7	18.1	17	31.4	33.8	49.5
SC205	19	1.2	.9	3	.3	.6	1.2
SCRS8	4	8.8	6.8	12	.3	.8	7.1
SCSD8	39	4.1	3.1	12	32.4	34.7	35.5
	20	6.8	5.1	12	14.9	17.7	20.0
	13	8.5	6.5	11	7.7	10.7	14.2
	8	10.1	7.7	12	4.4	6.5	12.1
	5	12.5	9.8	14	1.0	2.1	10.8
SCTAP1	10	1.9	1.5	6	.4	.9	1.9
	5	2.9	2.3	7	.4	.8	2.7

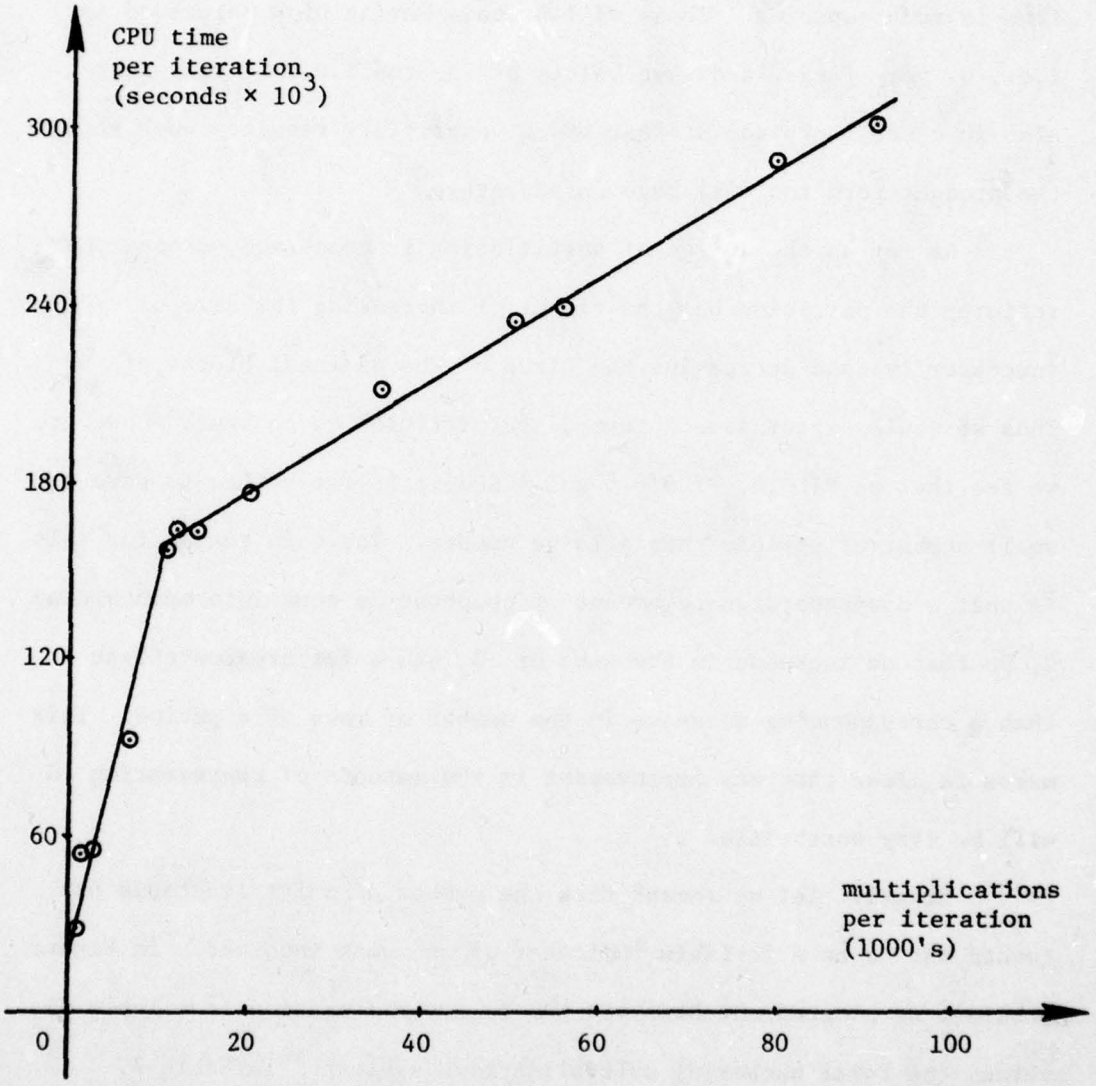
From this table, we see that with the exception of the PILOT8 run with $K = 8$, the explicit form for Q is always superior. However it should be noted that there will be models for which the product form is much superior. These will be ones having high values of p, i.e., G very large, and high values of ρ, the i.e., G very dense. Also in cases where the storage of Q explicitly requires much core, the product form too will have an advantage.

As far as the degree of partitioning is concerned, notice that refining the partition has the effect of increasing the size of G (necessarily) and decreasing the sizes of the diagonal blocks of $\bar{B}$. Thus we would expect some intermediate partition to be best. However, we see that on PILOT8, PILOT8.5 and SCSD8 it is far better to have a small number of periods than a large number. The main reason for this is that a disproportionate amount of computation goes into maintaining G, so that an increase in the size of G has a far greater effect than a corresponding increase in the number of rows of a period. This makes it clear that any improvement in the methods of representing G will be very worthwhile.

Finally, let us remark that the number of multiplications has turned out to be a reliable indicator of the work involved. In Figure 3 below, we plotted the observed CPU time per iteration (see Table 8) versus the total number of multiplications estimated in Table 7.

Figure 3

Graph of Time/Iteration Versus Multiplications/Iteration



With the exception of the 4 leftmost data points (corresponding to the small problems SC205, SCRS8, and SCTAP1) we see that this is a very good straight line fit.

9.2. Different Strategies During Update

So far we have considered the update and the subsequent analysis only from the point of view of maintaining the minimality of G at every step. Maintaining the minimality of G is expensive and should clearly only be done if the effect of not doing so is more expensive.

From the update flow diagram in Figure 1 we see that the points at which we do perform work to maintain minimality are the FIND2 and FINDB operations. Let us consider each of these in turn.

Recall that the FIND2 operation scans the columns of B (in some fixed period) that are not in $\bar{B}$ for a candidate to replace a column of $\bar{B}$ (in the same period) that is not in B. If there are ℓ columns of B in this period that are not in $\bar{B}$ then failure to find a suitable candidate costs us precisely ℓ TRANB operations. If we succeed in finding a candidate, the work involved is on the average $\ell/2$ TRANB operations and one each of the FCOL, INVUPD, DEL and RANK1 operations. What we gain is a G with its size reduced by one.

What will be of great importance here is the number of iterations remaining before we recompute the whole factorization and, in so doing, restore G to minimality. Denote this number by r. In order to make the following analysis possible we shall need to assume that if we do not reduce the size of G by one -- when it is possible -- the remaining r iterations will have a G that is always one larger than it would have been. In addition, we shall require an estimate of the probability of a FIND2 operation being successful. Denote it by θ. A very crude estimate of θ may be made by setting

$$\hat{\theta} = \frac{P_r\{\text{update combination 5}\}}{P_r\{\text{update combination 4}\} + P_r\{\text{update combination 5}\}}$$

where the probabilities on the right hand side are given in Table 11. Refer to Figure 1 and Table 2 for the meaning of "update combination i."

Now from the cross reference Table 3 we see that an FCOL operation requires $[K/2]$ TRANB operations, that an INVUPD operation requires half a TRANB operation, and that the DEL and RANK1 operations require 1.5 TRANQ's and 2.5 SWEEP's. Using arguments as in Section 8.1.1 this amounts to

$$([\tfrac{K}{2}] + \tfrac{1}{2})\eta + (1.5 + 15\rho)\, p^2$$

multiplications. Using a value of $\rho = .6$ and approximating $[K/2]$ by $K/2$ this reduces to

$$\tfrac{1}{2}(K + 1)\eta + 10.5\ p^2 \quad .$$

Let $W(p)$ denote the work per iteration (in multiplications) when G is $p \times p$. Then assuming that the size of G stays roughly constant, the expected net profit for a FIND2 operation is

$$\theta\{r[W(p) - W(p-1)] - \tfrac{1}{2}(K + \ell + 1)\eta - 10.5\ p^2\} - (1 - \theta)\ell\eta \quad .$$

From Table 11 we obtain for the explicit form, with $q = 30$,

$$W(p) - W(p-1) \approx \frac{1}{60} K \eta + \frac{2}{15} p^2 + 26\ p \ .$$

Taking as an example the PILOT8 model with $K = 4$, $p = 46$, $\eta = 1117$, $\ell = 12$ (obtained by guessing at $\ell = p/K$) and $\theta = .05$ (estimated as suggested above) we obtain the expected profit to be

$$14320 - 78r \ .$$

This is positive for $r > 184$.

We therefore conclude that with $p = 46$ and fewer than 184 iterations to go before a complete refactorization, it does not pay us to use the FIND2 operation. However on a problem like SCSD8 with $K = 39$, $p = 44$, $\eta = 26$, $\ell = 2$ (say), and $\theta = .02$ it turns out that the critical value of r is 17. Here it pays us to use the FIND2 operation most of the time.

With the FINDB operation, we may proceed similarly. Here we seek a column of B that is not in $\bar{B}$ to replace a given column of $\bar{B}$. Going back to the update flow diagram in Figure 1 we see that the FINDB operation occurs in three positions. For update combinations 2, 3 and 8, 9 we see that we either fail to find a suitable column and do no extra work, or we succeed and proceed with one each of the FCOL, INVUPD, DEL and RANK1 operations, exactly as in the FIND2 case. The

only difference between this and the FIND2 case is that whether we find a column or not, there is initially only one TRANB operation. Thus the expected net profit becomes

$$\theta\{r[W(p) - W(p-1)] - \tfrac{1}{2}(K+1)\eta - 10.5\ p^2\} - \eta \quad .$$

For the PILOT8 model with the same parameters as above, except for a new estimate of $\theta = .25$ (obtained as in the FIND2 case) we obtain the critical value of r to be 17. For the same SCSD8 model as before, we get $\theta = .38$ and a critical value of $r = 40$. The remaining FINDB operation in the flow diagram may be handled likewise.

Using ideas of this type for fine tuning will be valuable, especially on models where many runs are made, and reasonable estimates of the various parameters are available.

9.3. A Comparison with MINOS

We ran each of the problems on MINOS, adjusting tolerances and refactorization frequencies where necessary to obtain run times that were as fast as possible. The runs on LPBLK were all made with Q stored explicitly, and maintaining the minimality of G at each step. The following table resulted. (See over.)

ETA is the number of nonzeros in the representation of each $\bar{B}_{tt}$ for LPBLK and the whole basis for MINOS. The times are given in CPU milliseconds. All figures were computed as averages over the whole run.

Table 8

Name	K	LPBLK			MINOS		
		ETA(η)	Size of G (p)	Time/ Iteration	ETA	Spikes	Time/ Iteration
PILOT8	8 4	423 1117	71 46	302 240	14582	152	203
PILOT8.S	8 4	433 1168	66 42	288 235	15815	130	188
SC205	19	15	4	30	1021	23	25
SCRS8	4	502	4	92	3335	38	65
SCSD8	39 20 13 8 5	26 82 131 295 545	44 31 21 17 7	213 177 163 163 158	4358	56	130
SCTAP1	10 5	47 130	6 5	55 56	2204	28	40

From the CPU times in the above table, we see that MINOS is approximately 22% faster than LPBLK on PILOT8, PILOT8.S, SC205 and SCSD8. It is much faster on SCRS8 and SCTAP1.

Scaling seems to have had the same effect on both codes. In the case of MINOS the scaled columns would make the pivot elements of triangle columns more acceptable (in a relative test) and so force fewer triangle columns to become spikes. It would appear that a similar argument could be made for LPBLK: if more pivot elements become acceptable then the FINDB and FIND2 operations would be more successful in keeping the size of G to a minimum.

On the stability side, the only models to present problems were PILOT8 and PILOT8.S. In the runs recorded here, LPBLK experienced no trouble; however there were some advanced bases for PILOT8 that LPBLK failed to refactorize while MINOS succeeded. As mentioned in Section 7.2 the ratio

$$|r_{11}|/|r_{pp}|$$

is a lower bound on the condition number of G. On PILOT8 and PILOT8.S this ratio was typically of the order 10^8, sometimes going as high as 10^{10}. With the other models, this ratio was at most 10^3. The PILOT model has always been badly conditioned and a code like MPSIII [19] often needs to force unit columns into the basis and lose feasibility in order to maintain stability. MINOS is perhaps the only code that has not suffered unduly on PILOT.

One apparent cause of PILOT's illconditioning is the presence of dense 12 × 12 Leontief matrices imbedded in each period. The variables corresponding to these Leontief systems were all free[1] (no upper or lower bounds) and hence always in the basis. In the case of LPBLK, by the manner in which the $\bar{B}_{tt}$ are defined during a complete refactorization (see Section 7.3), the dense Leontief columns would be considered last for introduction into $\bar{B}$ and many of them would

[1] This was done since it was possible to demonstrate in advance that in any optimal solution, these activities would always be at a positive level.

not 'make it.' This would result in several columns of G being transforms of these Leontief columns, a fact that seemed intuitively wrong. Accordingly we forced all the Leontief columns into $\bar{B}$ by setting their "merit" counts artificially at zero. Having then chosen each $\bar{B}_{tt}$ in this way, we would refactorize the $\bar{B}_{tt}$'s immediately to achieve sparsity, after lifting the restrictions on the "merit" counts. The result of doing this was a much improved factorization: the lower bound on the condition number of G dropped by as much as two orders of magnitude.

9.4. Storage Considerations

Table 9 below contains statistics on the average number of nonzeros required to represent the basis factorization. The first two columns give the average number of nonzeros (for both LPBLK and MINOS) taken over the whole run.[1] The third column gives the average number of nonzeros recorded immediately after a MINOS refactorization.

This table shows that the basis storage requirements of LPBLK are between 23% and 60% of those of MINOS, a substantial savings.

The method of Bisschop and Meeraus [3] leaves the initial factorization of the basis untouched and instead updates a certain augmented matrix of size $k \times k$ where k is the number of columns in which the initial and current bases differ. A lower bound on their storage requirements is therefore given by the number of nonzeros in the

[1] These were taken directly from Table 8, with the entries in the LPBLK column being set equal to $K\eta + 3/2\ p^2$ for the explicit form of Q.

initial representation. Assuming that their initial factorization is an LU as performed by MINOS we may use the third column of Table 9 as the lower bound. Comparing this with the column for LPBLK, we observe, at the appropriate degree of partitioning that LPBLK is far superior on all except SCRS8 and SCSD8 where it is no worse that the lower bound.

Table 9

Storage Requirements

Name	K	Average Over The Run		Average at refactorization (MINOS)
		LPBLK	MINOS	
PILOT8	8 4	10946 7642	14582	11874
PILOT8.S	8 4	9998 7318	15815	12088
SC205	19	309	1021	641
SCRS8	4	2032	3335	2040
SCSD8	39 20 13 8 5	3918 3082 2365 2794 2798	4358	2331
SCTAP1	10 5	524 688	2204	1690

Another possibility is to use the factorization of LPBLK as the initial factorization in the Bisschop-Meeraus scheme. However, while this will more or less equalize the storage requirements it seems very unattractive from a stability point of view.

9.5. Bumps and Spikes

It is interesting to compare the bump and spike structure of a basis with its natural block triangular structure. Perhaps surprisingly the two appear to have little in common.

Table 10 below gives the bump and spike structure of a typical PILOT8 basis (626 × 626). This was found using Hellerman and Rarick's p^4 algorithm as implemented in MINOS.

Table 10

Bump and Spike Structure of a PILOT8 Basis

BUMP#	#COLUMNS	#SPIKES
1	2	1
2	3	2
3	2	1
4	2	1
5	2	1
6	2	1
7	2	1
8	2	1
9	2	1
10	2	1
11	13	8
12	381	100
13	3	2

Notice that there are several very small bumps and a single very large one. The block triangular structure on the other hand contains 8 blocks each having approximately 80 rows. Thus in such a case, a method like that of McBride [17] is unlikely to be effective.

Table 11

Probability Distribution of the 11 Update Combinations

Name	K	Update Combination										
		1	2	3	4	5	6	7	8	9	10	11
PILOT8	8	.073	.079	.012	.123	.003	.024	.243	.166	.032	.140	.107
	4	.057	.085	.013	.149	.008	.071	.221	.128	.057	.141	.071
PILOT8.S	8	.036	.083	.007	.186	.003	.036	.187	.202	.043	.117	.102
	4	.042	.072	.016	.263	.007	.061	.197	.117	.047	.097	.082
SC205	19	.000	.021	.025	.407	.004	.158	.112	.158	.066	.037	.012
SCRS8	4	.009	.012	.005	.599	.003	.055	.120	.071	.076	.024	.026
SCSD8	39	.002	.002	.000	.257	.004	.083	.172	.199	.122	.072	.087
	20	.000	.010	.004	.436	.006	.077	.129	.147	.097	.048	.044
	13	.000	.006	.004	.491	.024	.060	.130	.142	.078	.040	.031
	8	.000	.005	.002	.585	.013	.045	.107	.111	.064	.043	.025
	5	.000	.000	.000	.807	.101	.028	.046	.060	.038	.008	.004
SCTAP1	10	.006	.006	.009	.632	.000	.055	.133	.055	.073	.023	.009
	5	.006	.005	.008	.660	.000	.059	.125	.041	.074	.008	.013

Table 12

Average Number of Times an Operation is Used During Update

Name	K	Update Operation									
		INVUPD	FROW	FCOL	RSWAP	CSWAP	ADD	DEL	RANK1	FINDB	FIND2
PILOT8	8	.497	.680	.070	.287	.164	.243	.226	.497	.533	.126
	4	.553	.731	.149	.282	.212	.221	.206	.553	.511	.157
PILOT8.S	8	.620	.746	.089	.334	.153	.187	.190	.620	.524	.189
	4	.639	.779	.130	.251	.158	.197	.193	.639	.454	.270
SC205	19	.896	.950	.253	.270	.195	.112	.108	.896	.386	.411
SCRS8	4	.911	.941	.139	.163	.079	.120	.119	.911	.286	.603
SCSD8	39	.874	.839	.209	.323	.155	.172	.215	.874	.499	.261
	20	.909	.907	.183	.259	.126	.129	.151	.909	.394	.442
	13	.907	.929	.165	.229	.100	.126	.136	.907	.378	.515
	8	.909	.932	.123	.183	.088	.107	.104	.909	.302	.596
	5	.984	.988	.076	.098	.036	.046	.062	.984	.153	.817
SCTAP1	10	.905	.962	.136	.142	.078	.133	.096	.905	.275	.632
	5	.928	.972	.141	.128	.067	.125	.102	.928	.253	.660

Table 13

Name	K	ξ	μ	η	p	α	γ		ρ
							Explicit	Product	
PILOT8	8	.75	1.86	423	71	2.23	3.10	3.81	.55
	4	.88	2.60	1117	46	2.30	3.19	3.89	.74
PILOT8.S	8	.83	2.42	433	66	2.46	3.13	3.83	.47
	4	.91	3.65	1168	42	2.35	3.17	3.79	.66
SC205	19	1.20	.93	15	4	2.70	3.35	3.88	.95
SCRS8	4	1.08	1.30	502	4	2.49	3.21	3.59	.38
SCSD8	39	1.05	1.23	26	44	2.97	3.39	4.11	.52
	20	1.09	1.54	82	31	2.75	3.30	3.85	.51
	13	1.07	1.67	155	21	2.66	3.24	3.73	.61
	8	1.06	2.03	295	17	2.49	3.20	3.58	.50
	5	1.06	1.79	545	7	2.30	3.11	3.30	.55
SCTAP1	10	1.10	1.09	47	6	2.41	3.20	3.53	.42
	5	1.11	1.35	130	5	2.42	3.20	3.52	.51

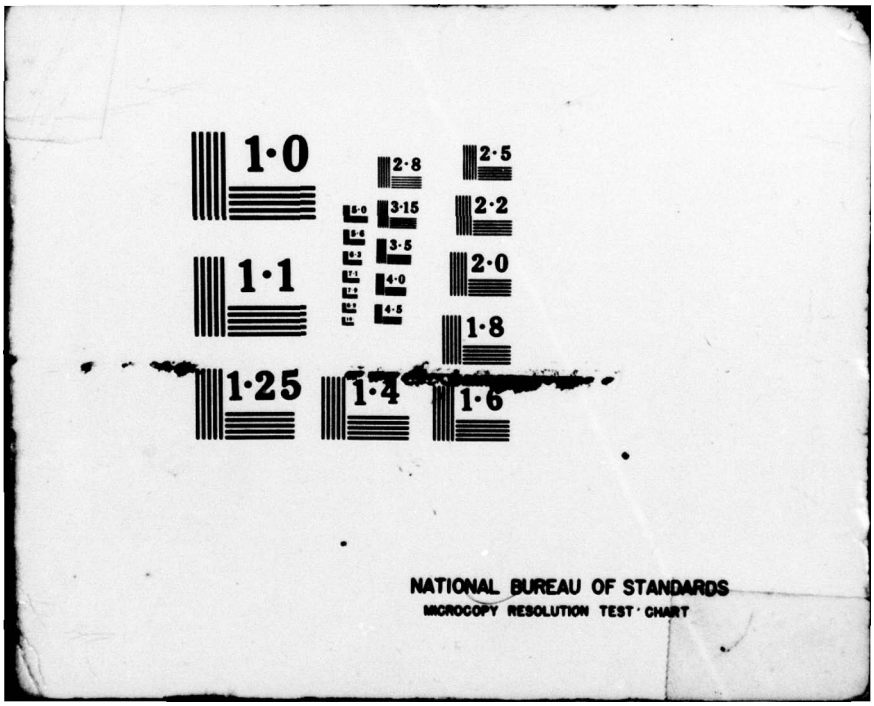
Key to Table 13

K : number of periods
ξ : average number of times per iteration that an FROW or FCOL operation is used
μ : average number of TRANB operations per iteration other than those in FROW and FCOL
η : average number of nonzeros in the representation of a typical $\bar{B}_{tt}$.
p : average size of G
α : average number of SWEEP operations per iteration
γ : average number of TRANQ operations per iteration
ρ : average proportion of rotations formed per sweep.

AD-A060 976 STANFORD UNIV CALIF SYSTEMS OPTIMIZATION LAB F/G 12/1
A BASIS FACTORIZATION METHOD FOR BLOCK TRIANGULAR LINEAR PROGRA--ETC(U)
APR 78 A F PEROLD, G B DANTZIG N00014-75-C-0267
UNCLASSIFIED SOL-78-7 NL

2 OF 2
ADA
060976

END
DATE
FILMED
1 -79
DDC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

10. Conclusion

As matters stand, the computational results of the previous section show that LPBLK saves considerably on storage requirements. However from the run times it is apparent that LPBLK comes close to, but is not yet competitive with the best available software. It seems that problems on which it will excel will be ones like PILOT8 with more periods and perhaps a greater proportion of non-zeros in the lower block triangle.

However the preceding analysis clearly shows that cheaper ways have to be found to deal with G. Any method that can halve the work involved with G will make a 30% difference in the run time, so making it very competitive on the test problems presented. One possibility currently under study is to use the factorization $Q\,G^T = R$ but then discarding Q altogether and instead maintaining G explicitly. To solve the equations $Gx = g$ we would first solve two triangular systems $R^TRy = g$, and then set $x = G^Ty$. Since G is typically between 30% and 50% dense it may well be possible to store it as a sparse matrix, so that forming G^Ty will be inexpensive. Thus solving equations this way will involve no more work than when we keep Q. Where we save considerably is in the update, since now we need only apply the rotations to R.

This approach was first used by Gill and Murray [10], in the context of the simplex method for dense systems. Paige [21] has shown that the above method of solving the equations is stable, involving only $K(G)$ (instead of $K^2(G)$) in the error bound. However,

to solve for the price vector, we need to solve equations of the form $G^T x = g$ (refer to Section 5.1). This requires the solution of the equations $R^T R x = Gg$ where now the term $K^2(G)$ does enter into the error bound. Since the price vector only indicates which column to bring into the basis, and is not used in any further computation, less stability here can be tolerated. Further, in Phase II we can simplify the procedure since we always solve $G^T x = e_r$ where e_r is a fixed unit vector. Multiplying this relation on the left by Q yields $Rx = Qe_r$. Therefore by maintaining the r^{th} column of Q explicitly from one iteration to the next, we can solve these equations by means of a single triangular system. This is both stable and inexpensive.

In addition to the choice of representation for G, the fine tuning aspects of Section 9.2 have yet to be properly implemented and tested. Some experimental runs made along these lines have been very encouraging.

More work needs also to be done on the stability side in general. There is a definite need for more care in deciding which columns go into $\bar{B}$ and which are transformed to become columns of G.

Acknowledgments

The authors are grateful to Michael Saunders, John Tomlin and Margaret Wright for the many hours of discussion and deliberation, and also for the use of codes they had written. Without them this work would not have been possible.

Thanks are also due to James Ho for supplying the SC205, SCRS8, SCSD8 and SCTAP1 test problems.

The authors wish to express special thanks to Michael Saunders for his careful reading of the manuscript and helpful suggestions.

References

[1] E.M.L. Beale, "Sparseness in Linear Programming" in Large Sparse Sets of Linear Equations, J.K. Reid (ed.) (1971), Academic Press London, pp. 1-15.

[2] E.M.L. Beale, "Advanced Algorithmic Features for General Mathematical Programming Systems" in Integer and Nonlinear Programming, J. Abadie (ed.) (1971), North-Holland Publishing Company, London.

[3] J. Bisschop and A. Meeraus, "Matrix Augmentation and Partitioning in the Updating of the Basis Inverse" Mathematical Programming, 13, 3 (1977), 241-254.

[4] A.J. Cooke and L.J. Shustek, "A User's Guide to MORTRAN2" Computation Research Group, Stanford Linear Accelerator, Stanford, California, CTGM No. 165, June 1975.

[5] G.B. Dantzig, "Upper Bounds, Secondary Constraints, and Block Triangularity in Linear Programming" Econometrica, 23, April 1955, pp. 174-183.

[6] G.B. Dantzig, Linear Programming and Extensions, (1963), Princeton University Press, Princeton, New Jersey.

[7] G.B. Dantzig, "Large-Scale Systems Optimization with Application to Energy" Technical Report SOL 77-3, April 1977, Department of Operations Research, Stanford University, Stanford, California.

[8] G.B. Dantzig and S.C. Parikh, "At the Interface of Modeling and Algorithms Research" Technical Report SOL 77-29, October 1977, Department of Operations Research, Stanford University, Stanford, California.

[9] J.J.H. Forrest and J.A. Tomlin, "Updating Triangular Factors of the Basis in the Product Form Simplex Method" Mathematical Programming, 2 (1972), 263-278.

[10] P.E. Gill and W. Murray, "A Numerically Stable Form of the Simplex Algorithm" Linear Algebra and its Applications, 7, (1973), 99-138.

[11] E. Hellerman and D. Rarick "The Partitioned Preassigned Pivot Procedure (P^4)" in Sparse Matrices and Their Applications, Eds., D.J. Rose and P.A. Willoughby, Plenum Press, New York, (1972), pp. 67-76.

[12] J.K. Ho and A.S. Manne, "Nested Decomposition for Dynamic Models" Mathematical Programming, 6 (1974) pp. 121-140.

[13] J.K. Ho, "Implementation and Application of a Nested Decomposition Algorithm" Proceedings of the Bicentennial Conference on Mathematical Programming, November - December 1976, Gaithersburg, Maryland.

[14] M. Kallio and E.L. Porteus, "Triangular Factorization and Generalized Upper Bounding Techniques" Operations Research, 25, 1, (1977) 89-99.

[15] C.L. Lawson and R.J. Hanson, Solving Least Squares Problems, Prentice-Hall, New Jersey, (1974).

[16] E. Loute, "A Revised Simplex Method for Block Structured Linear Programs" Doctoral Dissertation, Applied Sciences Faculty, Catholic University of Louvain, Belgium, 1976.

[17] R.D. McBride, "A Bump Triangular Dynamic Factorization Algorithm for the Simplex Method" Working Paper No. 19, December 1977, University of Southern California School of Business.

[18] MPS/360, Linear and Separable Programming User's Manual, IBM publication H20-0476.

[19] MPS III Users Manual (1973), Management Science Systems, Rockville, MD.

[20] B.A. Murtagh and M.A. Saunders, "A Large-Scale Nonlinear Programming System (for Problems with Linear Constraints) -- User's Guide" Technical Report SOL 77-9, February 1977, Department of Operations Research, Stanford University, Stanford, California.

[21] C. Paige, "An Error Analysis of a Method for Solving Matrix Equations" Math. Comp., 27, 122, April 1973, 355-359.

[22] A.F. Perold, "Continuous Linear Programming" Ph.D. Thesis (in preparation), Department of Operations Research, Stanford University, Stanford, California.

[23] M.A. Saunders, "A Fast, Stable Implementation of the Simplex Method Using Bartels-Golub Updating" in Sparse Matrix Computations, ed., J.R. Bunch and D.J. Rose, Academic Press, New York, New York, (1976), pp. 213-226.

[24] M.A. Saunders, "MINOS System Manual" Technical Report SOL 77-31, December 1977, Department of Operations Research, Stanford University, Stanford, California.

[25] J.A. Tomlin, "Pivoting for Size and Sparsity in Linear Programming Inve-sion Routines" J. Inst. Maths. Applics. (1972), 10, 289-295.

[26] J.A. Tomlin, "An Accuracy Test for Updating Triangular Factors" Mathematical Programming Study 4, (1975), 142-145.

[27] J.A. Tomlin, "On Scaling Linear Programming Problems" Mathematical Programming Study 4, (1975), 146-166.

[28] J.A. Tomlin, "LPM1 Users Guide" unpublished communication.

[29] R.D. Wollmer, "A Substitute Inverse for the Basis of a Staircase Structure Linear Program" Mathematics of Operations Research, 2, 3, (1977), 230-239.

[30] Margaret H. Wright and Steven C. Glassman, "Fortran Subroutines to Solve the Linear Least-Squares Problem and Compute the Complete Orthogonal Factorization" Technical Report SOL 78-8, April 1978, Department of Operations Research, Stanford University, Stanford, California.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (14) SOL-78-7

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle) (6)

A Basis Factorization Method for Block Triangular Linear Programs.

5. TYPE OF REPORT & PERIOD COVERED (9)

Technical Report

6. PERFORMING ORG. REPORT NUMBER

SOL 78-7

7. AUTHOR(s) (10)

Andre F. Perold and George B. Dantzig

8. CONTRACT OR GRANT NUMBER(s) (15)

N00014-75-C-0267

EY-76-S-03-0326-PA-18

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Department of Operations Research -- SOL
Stanford University
Stanford, CA 94305

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

NR-047-064

11. CONTROLLING OFFICE NAME AND ADDRESS

Operations Research Program -- ONR
Department of the Navy
800 N. Quincy Street, Arlington, VA 22217

12. REPORT DATE (11)

April 1978

13. NUMBER OF PAGES

98

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) (12) 103p.

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Large Scale Linear Programming
Time-Staged Linear Programs
Multi-Staged Linear Programs
Matrix Factorization and Updating
Block Triangular
Staircase
Persistence

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See Attached

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408765

A BASIS FACTORIZATION METHOD FOR BLOCK TRIANGULAR LINEAR PROGRAMS

by Andre F. Perold and George B. Dantzig

SOL 78-7

Time-staged and multi-staged linear programs usually have a structure that is block triangular. Basic solutions to such problems typically have the property that similar type activities persist in the basis over several consecutive time-periods. When this occurs the basis is close to being square block triangular. In 1955 Dantzig suggested a way of factorizing the basis to take advantage of this property. This paper discusses persistence in staircase models and then presents a considerable refinement of the above factorization algorithm.

The method has been implemented in an experimental code, with use being made of LU and QR factorization and updating techniques for the solution of small sub-systems of equations. An in-depth analysis is made of the work involved, and computational experience on several dynamic models is reported.