Mathematical Programming 23 (1982) 274–313.
North-Holland Publishing Company

SOLVING STAIRCASE LINEAR PROGRAMS BY THE SIMPLEX METHOD, 1: INVERSION

Robert FOURER

Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60201, U.S.A.

Received 29 January 1980
Revised Manuscript received 17 September 1981

This and a companion paper consider how current implementations of the simplex method may be adapted to better solve linear programs that have a staged, or 'staircase', structure. The present paper looks at 'inversion' routines within the simplex method, particularly those for sparse triangular factorization of a basis by Gaussian elimination and for solution of triangular linear systems. The succeeding paper examines 'pricing' routines. Both papers describe extensive (though preliminary) computational experience, and can point to some quite promising results.

Key words: Large-Scale Optimization, Linear Programming, Staircase Linear Programs, Simplex Method.

1. Introduction

Staircase-structured linear programs (LPs) have been studied about as long as linear programming itself. They typically arise in economic planning over time: activities $x_t = (x_{t1}, \ldots, x_{tn})$ are run in a series of periods $t = 1, \ldots, T$, subject to constraints that link activities in adjacent periods. The resulting LPs, in their simplest form, have a structure like this:

$$\begin{array}{lrl}
\text{maximize} & c_1x_1 + c_2x_2 + c_3x_3 + \cdots + c_{T-1}x_{T-1} + c_Tx_T, & \\
\text{subject to} & A_{11}x_1 & = b_1, \\
& A_{21}x_1 + A_{22}x_2 & = b_2, \\
& A_{32}x_2 + A_{33}x_3 & = b_3, \\
& \cdots & \\
& \cdots \cdot A_{T,T-1}x_{T-1} + A_{T,T}x_T & = b_T, \\
& x_1, \ldots, x_T \geq 0. &
\end{array}$$

The submatrices of coefficients, A_{tt} and $A_{t+1,t}$, form the 'staircase'.

In the infancy of computers this sort of structured problem was attractive because it seemed to offer a hope of solving practical LPs in a reasonable amount of time. Thus in 1949 Dantzig [7] observed that:

> "... while the general mathematical problem is concerned with maximization of a linear form of nonnegative variables subject to a system of linear equalities, in the linear programming case

> one finds by observing the above [staircase] system that the grand matrix of coefficients is composed mostly of blocks of zeroes except for submatrices along and just off the 'diagonal'. Thus any good computational technique for solving programs would probably take advantage of this fact."

At the time of this quotation, computers were primitive; the simplex method could be costly and time-consuming even for small problems of general structure. Hence staircase-structured LPs were of interest not only for their applications, but for the possibility that much faster versions of the simplex method could be devised to solve them.

Staircase linear programs are of no less interest today. Along with economic planning, they have found applications in production scheduling, inventory, transportation, control, and design of multistage structures [34, 38]. Yet a recent survey [22] observes that:

> "the 'staircase' model, in which similar sets of variables and constraints are replicated many times, seems no more tractable today than when its importance was recognized over 20 years ago. . . . Today we know only how to solve it as we would any linear programming problem; but this type of problem requires more work to solve than does the average problem of the same size. However, there should be some way to take advantage of its simple structure."

Thus the situation has been reversed. The general simplex method is now impressively fast rather than impossibly slow, while staircase LPs are a troublesomely hard case rather than a promisingly easy one.

1.1. Proposed methods for staircase LPs

There has certainly been no shortage of attempts to solve staircase LPs more efficiently. Although the simplex method has usually been involved in some guise, individual proposals have varied considerably. The essential ideas of these proposals may be classified in four broad areas:

Compact basis methods employ a special representation of the basis or basis inverse in conjunction with a more or less standard simplex method. This approach was first suggested by Dantzig [8, 10], and early variations were employed by Heesterman and Sandee [27] and Saigal [52]. More recent compact-basis schemes have been worked out by Dantzig [11], Wollmer [58], Marsten and Shepardson [41], Perold and Dantzig [48], Propoi and Krivonozhko [49], Bisschop and Meeraus [5], and Loute [33].

Nested decomposition methods apply the Dantzig–Wolfe decomposition principle to generate a series of sub-problems at each period. This approach was suggested by Dantzig and Wolfe in their original paper on decomposition [12], and has been extended or modified by Cobb and Cord [6], Glassey [23, 24], and Ho and Manne [35].

Transformation methods start with a simpler LP that can be solved easily, and work toward a solution of the original staircase LP. Varied proposals in this class are from Grinold [26], Aonuma [1], and Marsten and Shepardson [41].

Continuous methods deal with a multi-period LP in continuous rather than discrete time. Fundamentals of a simplex method for continuous-time linear programming have been proposed by Perold [47].

Ho and Loute have reported promising experiments with their methods [33], but computational experience with most proposals is negligible. At present no method has been shown to be more effective than the general simplex method in solving a wide variety of large staircase problems.

1.2. Adaptation of the simplex method to staircase LPs

In contrast to proposals for staircase methods, proposals for improving the general simplex method have been quite successful. The simplex method has consequently developed into an amalgam of fairly sophisticated algorithms, many of which are objects of study in their own right and are not normally thought of in connection with linear programming. As a result, the simplex method has become more and more a specialist's domain.

It is therefore not surprising that study of staircase LPs has tended to diverge from study of the simplex method. Staircase linear programming, typified by the above-listed papers, has sought staircase methods to replace the original simplex method; in the mean time new, better simplex techniques have emerged for general linear programming, but have not been applied to special structures such as staircases.

This paper and its successor [20] seek to reverse the trend; they are concerned with adapting the modern simplex method to solve staircase LPs more efficiently. Each paper looks at a set of algorithms within the simplex method: this one deals with 'inversion' of the basis—more accurately, solution of linear systems by Gaussian elimination—and the succeeding one considers partial pricing.

Both papers describe extensive, although preliminary, computational experience. The results are quite promising: a staircase-adapted simplex method sometimes performs considerably better than the general method, yet on a range of large problems it is never significantly worse. Moreover, it is possible to identify several promising opportunities for further improvement.

1.3. Outline of this paper

The first two sections below develop the terminology and properties employed in studying linear systems that arise from staircase linear programs. Section 2 covers staircase LPs and pertinent features of staircase linear systems. Section 3 looks at the particular kinds of linear systems that must be solved in the simplex method.

The following sections examine how the simplex method might better solve linear systems for staircase LPs. Section 4 describes sparse triangular fac-

torization of a staircase basis matrix, and Section 5 examines in detail various staircase solution algorithms for triangular systems.

Finally, Section 6 presents the results of preliminary but substantial computational experiments on a set of practical test problems. Detailed timings are employed in this section to compare staircase and non-staircase versions of a simplex-method code; an informal comparison with a commercially-distributed LP code is also reported. Implications of these experiments for future implementations are discussed briefly in Section 7's concluding remarks.

2. Staircase linear programs

Staircase linear programs share two simple characteristics: their variables fall into some sequence of disjoint groups, and their constraints relate only variables within adjacent groups. Usually the sequence of groups corresponds to a sequence of times, so that variables of a group represent activities during one time *period.* Constraints thus indicate how activities of one period are related to activities of the next.

A variable of period t will be called a *period-t variable.* By analogy, a constraint that involves variables of period t but not of later periods will be referred to as a constraint of period t, or as a *period-t constraint.*

Typically some period-t constraints involve only variables of period t, while others relate variables of periods t and $t-1$; the latter are said to be *linking* constraints, whereas the former are *non-linking.* Analogously, period-t variables that appear in constraints of period t and $t+1$ are linking variables, whereas variables that appear only in the constraints of period t are non-linking.

2.1. Staircase LPs of higher orders

A more general approach says that a staircase linear program is *of order p* if its constraints relate variables that are at most p periods apart. The preceding definitions thus characterize staircase LPs of order one. Higher-order staircase LPs are not uncommon in complex applications (for example, modeling energy systems [46]).

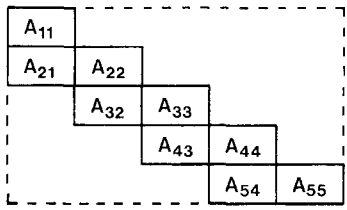
This paper is predominantly concerned with first-order staircase LPs, which have the most specialized structure and are consequently most amenable to special techniques. The adjective 'first-order' will therefore generally be omitted. Nevertheless, many techniques in this paper are essentially applicable to higher-order staircases as well, with appropriate modifications that will be pointed out as the exposition proceeds.

Higher-order staircase LPs can also be made into first-order ones, in either of two ways. First, pth-order equations can be transformed to equivalent first-order ones by adding certain variables and constraints. This yields a larger first-order

LP that has the same number of periods. Second, every p periods of the pth-order LP may simply be aggregated as one period. The result is a first-order staircase LP of the same size but having only about $1/p$ as many periods. The first method is most practical when the LP is nearly first-order to begin with, while the second may be feasible when the number of periods is large relative to p.

2.2. Staircase matrices

The matrix of constraint coefficients of a staircase linear program is a *staircase matrix*. Its nonzero elements are confined to certain submatrices centered roughly on and just off the diagonal:



Formally, a staircase structure for an $m \times n$ matrix A is defined as follows. Partition the rows into T disjoint subsets, and the columns into T disjoint subsets, so that the matrix is partitioned into T^2 submatrices, or 'blocks':

$$A_{ij} = \begin{bmatrix} \text{all elements in the } i\text{th row partition} \\ \text{and } j\text{th column partition of } A \end{bmatrix}, \quad i, j = 1, \ldots, T.$$

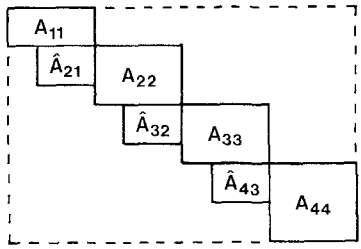
A is *lower staircase* (as drawn above) if $A_{ij} = 0$ except for $i = j$ and $i = j + 1$. A is *upper staircase* if $A_{ij} = 0$ except for $i = j$ and $i = j - 1$. Any upper-staircase matrix may be permuted to lower-staircase form by reversing the order of the periods [17, 19]. Hence it suffices to consider matrices A that have a specified lower-staircase structure, and hereafter 'staircase' will be used synonymously with 'lower staircase'.

By analogy with staircase linear programs, rows in the ith partition of a staircase matrix A are called period-i rows, and columns in the jth partition are called period-j columns. If a period-i row has nonzero elements in blocks $A_{i,i-1}$ and A_{ii}, it is a linking row; if it has non-zeroes only in A_{ii} it is a non-linking row. Similarly, a period-j column that has nonzeroes in A_{jj} and $A_{j+1,j}$ is a linking column, whereas one that has nonzeroes in A_{jj} only is a non-linking column.

If a period-i row is entirely zero within A_{ii}, that row may be moved back to period $i - 1$ without disrupting the staircase structure; analogously, a period-j column that is all-zero within A_{jj} may be moved to period $j + 1$. Nothing is lost, therefore, in assuming that the diagonal blocks A_{tt} have no all-zero rows or columns; A is then said to be in *standard* staircase form [17, 19]. Henceforth it will be assumed that all staircase LPs have a constraint matrix A in this standard

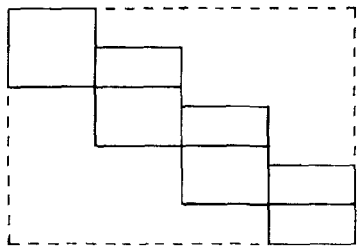
form. (The trivial case in which A has an all-zero row or column is thus ruled out.)

Following [17, 19], the period-i rows may be permuted to put the linking rows first, and the period-j columns may be permuted to put the linking columns last. Then A has the *reduced* form:



The reduced block $\hat{A}_{t,t-1}$ is just the intersection of the period-t linking rows and the period-$(t-1)$ linking columns.

If the linking rows of every period i are switched to period $i-1$, then A gains an alternative *row-upper-staircase* form:



Switching the linking columns of period j to period $j+1$ gives a different, *column-upper-staircase* form. Thus a staircase A in reduced standard form embodies three staircases—lower, row-upper, and column-upper—each corresponding to a different choice of where the periods begin and end.

2.3. Staircase bases

Any basis B of a staircase linear program necessarily inherits a staircase structure from the constraint matrix A; B's staircase blocks, $B_{t,t-1}$ and B_{tt}, may be taken to be the sub-blocks of $A_{t,t-1}$ and A_{tt} that contain only the basic columns. If A has a reduced form, $\hat{B}_{t,t-1}$ may likewise be taken as the basic part of $\hat{A}_{t,t-1}$.

The inherited staircase of B need not be in standard or reduced form, even though A is. Specifically, either B_{tt} or $\hat{B}_{t,t-1}$ may be zero along some linking row i, if it happens that, in A_{tt} or $\hat{A}_{t,t-1}$, all the nonzeroes along row i are in non-basic

columns. In this event B may be returned to reduced standard form by reassigning certain rows and columns. Any linking row that is zero in B_{tt} becomes a non-linking row in period $t-1$; in the process, some linking columns of period $t-1$ may become non-linking. Any linking row that is zero in $B_{t,t-1}$ becomes a non-linking row.

It is generally most convenient to deal with B in its inherited staircase form, whether standard, reduced or neither. However, better results may be achieved by using B's reduced standard form instead, especially as it has fewer linking rows and columns and hence a tighter structure. This issue is considered further in Section 5.

Henceforth B_{tt} and $B_{t,t-1}$ (or $\hat{B}_{t,t-1}$) will represent the blocks of B's chosen staircase form, whether inherited or reduced standard. The number of rows in period i will be denoted m_i, and the number of columns in period j will be n_j; the respective numbers of linking rows and columns will be $\hat{m}_i$ and $\hat{n}_j$. For the row-upper-staircase form, the number of rows in period i will be m^i, and for the column-upper-staircase form the number of columns in period j will be n^j. Necessarily $\Sigma m_i = \Sigma m^i = \Sigma n_j = \Sigma n^j = m$, and $\hat{m}_i \leqslant m_i$, $\hat{n}_j \leqslant n_j$.

2.4. Balance constraints and square sub-staircases

If the staircase LP has a special dynamic Leontief structure [9], then in each period the number of basic columns must exactly equal the number of rows: $n_t = m_t$ for all t, and all blocks B_{tt} are square. This is not the case in general, however. A basis B of an arbitrary staircase LP may have $n_t > m_t$ for some periods t and $n_t < m_t$ for others.

Since the basis is nonsingular, however, it must obey the 'balance constraints' developed in [17, 19]. In summary, these restrict the excess of basic columns over rows in each period, individually and cumulatively, as folows:

$$0 \leqslant \sum_{1}^{t} (n_i - m_i) \leqslant \min(\hat{m}_{t+1}, \hat{n}_t), \qquad t = 1, \ldots, T-1,$$

$$-\min(\hat{m}_s, \hat{n}_{s-1}) \leqslant \sum_{s}^{t} (n_i - m_i) \leqslant \min(\hat{m}_{t+1}, \hat{n}_t), \quad s, t = 2, \ldots, T-1,$$

$$-\min(\hat{m}_s, \hat{n}_{s-1}) \leqslant \sum_{s}^{T} (n_i - m_i) \leqslant 0, \qquad s = 2, \ldots, T.$$

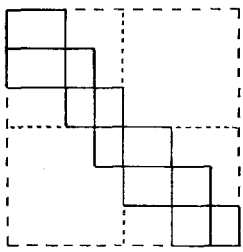
In words, the cumulative imbalance between rows and basic columns in periods s through t is bounded by the smaller dimension of $\hat{B}_{s,s-1}$ and the smaller dimension of $\hat{B}_{t+1,t}$. Hence these constraints are quite strict when there are relatively few linking rows or columns.

The first constraint above may also be written as the following three inequalities:

$$\sum_{1}^{t} n_i \geqslant \sum_{1}^{t} m_i, \qquad \sum_{1}^{t} n_i \leqslant \sum_{1}^{t} m^i, \qquad \sum_{1}^{t} n^i \leqslant \sum_{1}^{t} m_i.$$

These say that the first t periods of the lower staircase cannot have more rows than columns, while the first t periods of the associated row-upper or column-upper staircase cannot have more columns than rows.

All three of these relations are equalities when $t = T$, since B is square. It can also happen that equality is achieved for some $t < T$. For example, if $\sum_1^t m_i = \sum_1^t n_i$, B must look something like this:



$$\sum_1^3 m_i = \sum_1^3 n_i.$$

The rows and columns of periods 1 through t from a *square sub-staircase*, as do the rows and columns of periods $t+1$ through T; they are linked only by nonzero elements in the off-diagonal block $\hat{B}_{t+1,t}$. In a similar way an equality $\sum_1^t n_i = \sum_1^t m^i$ implies a pair of square sub-staircases within the row-upper staircase form, and $\sum_1^t n^i = \sum_1^t m_i$ implies the same for the column-upper form.

Generally, B may exhibit any or all of these three kinds of equalities, and each may hold for several values of $t < T$. If p different such equalities hold, then B breaks into $p + 1$ disjoint square sub-staircases of various kinds. The presence or absence of sub-staircases will be of importance to several of the techniques described further on in this paper.

3. Solving linear systems in the simplex method

In solving linear programs by the simplex method, a great deal of computational effort is devoted to 'inverting the basis'. More precisely, at each iteration the simplex method solves a linear system in B and a linear system in B^T, where B is an $m \times m$ basis matrix of columns from the constraint matrix A. Different realizations of the simplex method may construct and use these two linear systems in different ways (see [25] for example) but no practical version of the method avoids solving some system in B and some in B^T.

The preferred approach to solving linear systems in B and B^T—especially when B is very sparse and possibly ill-conditioned, as in the case of staircase LPs—is based on Gaussian elimination. Fundamentally, this approach computes a factorization of the form

$$B = LU$$

where L is lower-triangular and U is upper-triangular. (See [4, 50] for comparisons with other approaches.)

This section begins by reviewing aspects of LU computation, organization, and updating. It then looks more closely at the work of solving (fully or partially) the triangular system involving L or U, as efficient solution of these systems will be a key to more efficient handling of staircase LPs.

3.1. Computing an LU factorization of the basis

For the sake of concreteness, denote the two linear systems of the simplex method by

$$By = a,$$
$$B^{\mathrm{T}}\pi = f,$$

where the right-hand-side constants a and f are appropriately chosen. Since the order of variables and equations is arbitrary, the rows and columns of B may be permuted in any way in the course of solving these systems. In other words, for any permutation matrices P and Q, it suffices to solve

$$(PBQ^{\mathrm{T}})(Qy) = (Pa),$$
$$(QB^{\mathrm{T}}P^{\mathrm{T}})(P\pi) = (Qf),$$

which may be viewed as systems in the permuted matrix PBQ^{T} and its transpose.

Given a factorization $PBQ^{\mathrm{T}} = LU$, the above systems reduce to

$$L(U[Qy]) = [Pa],$$
$$U^{\mathrm{T}}(L^{\mathrm{T}}[P\pi]) = [Qf].$$

The system in B is thus transformed to two much easier systems: a lower-triangular one in L and an upper-triangular one in U. Likewise, the system in B^{T} is transformed to a lower-triangular one in U^{T} and an upper-triangular one in L^{T}.

The 'hard' part of solving $By = a$ or $B^{\mathrm{T}}\pi = f$ is thus the computation of $PBQ^{\mathrm{T}} = LU$ by Gaussian elimination. The essential operations of this computation are defined by the following recursion:

$$\beta^{(1)} = PBQ^{\mathrm{T}},$$
$$\beta_{ij}^{(k+1)} = \beta_{ij}^{(k)} - \beta_{ik}^{(k)}\beta_{kj}^{(k)}/\beta_{kk}^{(k)}, \quad i, j = k+1, \ldots, m;\ k = 1, \ldots, m-1,$$

of which L and U are a by-product:

$$\left.\begin{aligned} L_{ik} &= \beta_{ik}^{(k)}/\beta_{kk}^{(k)}, & i &= k, \ldots, m \\ U_{kj} &= \beta_{kj}^{(k)}, & j &= k, \ldots, m \end{aligned}\right\} \quad k = 1, \ldots, m.$$

The 'pivot' values $\beta_{kk}^{(k)}$ are critical to this procedure. An LU factorization exists if and only if all pivots are nonzero, and is numerically stable only if all pivots are sufficiently large in magnitude, both absolutely and relative to other elements of $\beta^{(k)}$.

Practical Gaussian elimination thus looks for permutations P and Q^{T} such that

PBQ^T has an acceptably large series of pivots. Generally, any initial choice of P and Q—the initial 'pivot selection'—may have to be modified as the recursion is carried out, in order to produce acceptable values of $\beta_{kk}^{(k)}$. Such a modification—so-called 'dynamic pivot selection'—may be made at any step k without affecting the computations at preceding steps.

Choice of P and Q also strongly affects the sparsity of nonzero elements in the resulting L and U, as have been shown both theoretically and experimentally [13, 15]. A good choice of pivots can assure that the number of nonzeroes in L and U is not much greater than the number in B, without sacrificing numerical stability; it is this property that makes LU factorization preferable in linear programming, where B is typically less than 1% dense. Section 4 will consider both initial and dynamic pivot selections that are particularly useful for sparse staircase LPs.

3.2. *Organizing the LU factorization of the basis*

The defining recursion of Gaussian elimination does not entirely fix the order in which the operations are carried out. Consequently there is some leeway in choosing the order in which the elements of L and U are computed and stored. In practice, this order is most strongly influenced by the way that storage is arranged.

The specifics of the simplex method greatly favor storage of the coefficient matrix A by column. Consequently, LP storage schemes invariably make it easy to retrieve the nonzero elements of any column of A; retrieving the elements of a row of A is much more difficult. Since any basis B is just a subset of the columns of A, it inherits A's storage scheme and has the same retrieval properties.

Because B is stored column-wise, Gaussian elimination for linear programming is most often arranged so that it processes only one column of B at a time. In outline, this form of elimination proceeds as follows:

FACTOR-BY-COLUMN:
1: SET $L = U = I$.
2: REPEAT for each column b_k of BQ^T, $k = 1, \ldots, m$:
  2.1: SOLVE $Lx = Pb_k$ for x,
  2.2: SET $U_{ik} = x_i$ for $i = 1, \ldots, k$,
  2.3: SET $L_{ik} = x_i/x_k$ for $i = k+1, \ldots, m$.

This algorithm produces both L and U one column at a time, and both are normally stored column-wise just as B is. It is quite straightforward to design an efficient and stable version that takes advantage of the sparsity of B, L and U. (Practical implementations also avoid performing any explicit divisions in recording L. However, the subsequent discussion assumes, for purposes of clarity, that the divisions are actually carried out.)

An alternative organization, employed by Reid [50, 51], follows the defining recursion much more closely. Its outline is as follows:

FACTOR-BY-ROW-AND-COLUMN:
1: SET BETA = PBQ^T.
2: REPEAT for $k = 1, \ldots, m$:
  2.1: SET $L_{ik} = \text{BETA}_{ik}/\text{BETA}_{kk}$ for $i = k, \ldots, m$,
  2.2: SET $U_{kj} = \text{BETA}_{kj}$ for $j = k, \ldots, m$,
  2.3: SET $\text{BETA}_{ij} = \text{BETA}_{ij} - \text{BETA}_{ik}\ \text{BETA}_{kj}/\text{BETA}_{kk}$ for all $i, j = k+1, \ldots, m$.

Here L is again produced by column but U is produced by row; in Reid's implementation, L is stored column-wise like B and U is stored so that it may be accessed readily by row or by column. Implementation of this arrangement is somewhat complex, requiring careful use of storage-management routines.

Choice of an elimination method and a storage scheme for L and U are important to staircase LPs in two respects. First, the storage scheme determines how linear systems involving L and U are solved (as explained later on in this section) and so influences the extent of savings to be expected in solving these systems for staircase LPs (Section 5). Second, different storage schemes are apprpriate to different methods for sparse staircase elimination (Section 4).

3.3. *Updating the LU factorization of the basis*

Typically a full LU factorization as described above is computed only every 50–100 iterations. At intervening iterations it is efficient to simply update the factorization, because the simplex method changes B by only one column at each iteration.

In general terms, an updating scheme starts with a factorization $P_0B_0Q_0^T = L_0U_0$ and derives, after l iterations, some factorization of $P_lB_lQ_l^T$ as a product of 'simple' matrices. For example, Benichou *et al.* [4] discuss a factorization of the form $P_0B_lQ_0^T = L_0U_0E_1E_2 \ldots E_l$ in which each E_i differs from the identity in only one column.

For staircase applications, however, the most appealing update scheme is of the kind originated by Bartels and Golub [2, 3]. In essence, Bartels–Golub updates determine a factorization of the form

$$P_0B_lQ_l^T = L_0(P_1^TL_1)(P_2^TL_2) \ldots (P_l^TL_l)U_l.$$

The rows of B_l are permuted like those of B_0, but the columns may be rearranged. The factors of the permuted B are the original lower-triangular matrix L_0, l additional permuted lower-triangular matrices $P_i^TL_i$, and a modified upper-triangular matrix U_l. Numerous detailed update schemes have been built on this idea, including those of Forrest and Tomlin [16], Reid [50, 51], Saunders [53, 54], and Gay [21].

A common feature of Bartels–Golub updates is that $L_1, \ldots, L_l$ all differ from the identity in fairly few elements, while U_l is not greatly changed from U_0. Thus the bulk of the work of solving a linear system, at any iteration, consists of solving one lower-triangular system and one upper-triangular system. Moreover, for a system in B_l the work always begins with solution of a system in L_0, and always ends with solution of a system (in U_l) that is like a system in U_0. Analogously, for a system in B_l^T the work always begins with solution of a system (in U_l^T) that is like a system in U_0^T, and always ends with solution of a system in L_0^T.

Thus little will be lost in the sequel by simply imagining the basis to be factored $PBQ^T = LU$ at each iteration. The complications introduced by updating—the insertion of simple factors between L and U, and the modification of U—will be mentioned only in the few instances where they make a difference.

3.4. *Solving triangular linear systems*

Linear systems in triangular form are solved by a simple, familiar, and numerically stable process of substitution. For a lower-triangular system $Lx = r$, this process works forward in L from L_{11} to L_{mm} and produces the solution vector one value at a time in the forward order $x_1, \ldots, x_m$. For an upper-triangular system $L^Tx = r$, on the other hand, substitution works backward from L_{mm} to L_{11} and produces the solution in the backward order $x_m, \ldots, x_1$. In an analogous fashion, solving an upper-triangular system in U also involves a backward substitution, whereas solving a lower-triangular system in U^T involves a forward substitution.

Following Saunders' terminology [55] a forward substitution will henceforth be referred to as an FTRAN, and a backward substitution as a BTRAN. Solving a system in L will be called an FTRANL, and solving a system in L^T will be called a BTRANL; solving a system in U will be a BTRANU and solving a system in U^T will be an FTRANU. Thus, for example, to solve $By = a$ given $PBQ^T = LU$, one applies an FTRANL and then a BTRANU to Pa, producing Qy in the order $(Qy)_m, \ldots, (Qy)_1$. To solve $B^T\pi = f$, one applies an FTRANU and then a BTRANL to Qf, producing $P\pi$ in the order $(P\pi)_m, \ldots, (P\pi)_1$.*

Even though BTRANs and FTRANs are simple in concept, they involve many operations for a large LP and may comprise a substantial proportion of the work in the simplex method. Two circumstances are crucial in determining the expense of a BTRAN or FTRAN routine for a given L or U: the storage organization of L or U, and the sparsity of the solution vector.

For purposes of illustration, suppose that the nonzero elements of a lower-triangular L are available in column-wise order. The essentials of an FTRANL

* This terminology is at variance with the traditional use of FTRAN and BTRAN to describe the solution of systems in B and B^T, respectively [16, 45]. In particular, the meanings of FTRANU and BTRANU in this paper are exactly the *reverse* of their meanings in the work of Benichou *et al.* [4].

routine to solve $Lx = r$ are as follows:

FTRANL:
REPEAT FOR j FROM 1 TO m:
SET $x_j = r_j/L_{jj}$,
REPEAT FOR i FROM $j+1$ TO m WHERE $L_{ij} \neq 0$:
SET $r_i = r_i - L_{ij}x_j$.

If $r_j = 0$ at the jth pass through the outer REPEAT loop, then also $x_j = 0$, and the inner loop merely adds zero to various elements of r. Hence the jth pass is superfluous when $x_j = 0$. Moreover, if it happens that $r_1, \ldots, r_{p-1}$ are all zero, then the main loop does no work until pass p. A more efficient routine is thus as follows:

FTRANL:
1: REPEAT (SET $x_p = 0$) FOR p FROM 1 UNTIL $r_p \neq 0$.
2: REPEAT FOR j FROM p TO m:
IF $r_j = 0$: SET $x_j = 0$,
ELSE: SET $x_j = r_j/L_{jj}$,
REPEAT FOR i FROM $j+1$ TO m WHERE $L_{ij} \neq 0$:
SET $r_i = r_i - L_{ij}x_j$.

Step 1 is especially valuable when $r_1, \ldots, r_{p-1}$ are known beforehand to be zero. The efficiency of step 2 depends on how sparse x turns out to be. If L and r are both sparse to begin with then x may well be fairly sparse.

The situation for L^T is quite different. Since L is stored column-wise, L^T is effectively stored row-wise, and a BTRANL routine for solving $L^Tx = r$ must proceed as follows:

BTRANL:
REPEAT FOR j FROM m TO 1:
REPEAT FOR i FROM m TO $j+1$ WHERE $L_{ij} \neq 0$:
SET $r_j = r_j - L_{ij}x_i$,
SET $x_j = r_j/L_{jj}$.

Here the jth pass cannot be avoided by knowing $r_j = 0$, since r_j is continually modified within the inner REPEAT loop and x_j is not determined until after the inner loop is completed. The only substantial economy from sparsity of r (or x) would be in knowing that *all* of $r_m, \ldots, r_{p+1}$ are zero; then $x_m, \ldots, x_{p+1}$ are also all zero and the outer loop may be started with $j = p$.

The key difference in the above examples (and their analogues for U) is in their storage organization. Briefly, column-wise organization is preferable to row-wise organization in taking advantage of zeroes within the right-hand-side

and solution vectors. Thus if L and U are both stored by column (the most common arrangement), then zeroes benefit systems in $B = LU$ much more than systems in $B^T = U^T L^T$. If there is also some access to U by row (as in the case with certain updating methods), then zeroes may also be taken into account in the FTRANU routines for solving systems in B^T.

The practical significance of the above remarks necessarily depends on the actual sparsity of the pertinent vectors. For systems of the form $By = a$, the vector a is always a column of the very sparse LP coefficient matrix A, and so FTRANL and BTRANU can indeed take advantage of considerable sparsity. For systems of the form $B^T\pi = f$ the situation is more involved (see Section 6), but generally BTRANL is most likely to suffer by being unable to take advantage of zeroes. These observations will be amplified in considering staircase LPs, whose sparsity structure is especially well defined.

3.5. Partially solving triangular linear systems

One consequence of the preceding analysis is that the solution to $By = a$ or $B^T\pi = f$ is ultimately computed one element at a time, regardless of how L and U are stored. The vector y is produced by BTRANU in the order $(Qy)_m, \ldots, (Qy)_1$, and the vector π is produced by BTRANL in the order $(P\pi)_m, \ldots, (P\pi)_1$.

BTRANL or BTRANU may therefore be terminated prematurely if only part of y or π needs to be computed. Such a partial solution has two potential uses: when the rest of the vector is already known (to be zero, for instance) and when only a portion of the vector is required at the present iteration.

Nevertheless, for general LPs there is little to be gained from trying to compute partial solutions, owing to the presence of the permutations P and Q. There is no efficient way, for example, to tell whether $(Qy)_j, \ldots, (Qy)_1$ will all be zero for some j, or to predict which element of $P\pi$ will be needed. Section 5 will show, however, that partial solutions *can* offer economies in solving staircase LPs, provided P and Q are chosen to reflect the staircase structure.

4. Sparse elimination of staircase bases

The staircase matrices encountered in linear programming are sparse in two senses; they have many blocks that are all-zero, and they have a low proportion of nonzero elements (typically 2–20%) within the remaining blocks. Thus staircase bases are prime candicates for the techniques of sparse Gaussian elimination. In essence, these techniques try to choose permutations P and Q and to factor $PBQ^T = LU$ so that the triangular matrices L and U are nearly as sparse as B.

Staircase sparse-elimination techniques use the staircase structure of B to

guide the choice of P and Q. Two families of such techniques have been proposed [17]; one is based on the 'bump-and-spike' sparse-elimination methods common in linear programming, and the other employs popular 'merit' methods of sparse elimination.

This section summarizes the direct effects—on speed, storage, and LU sparsity—of substituting staircase sparse-elimination techniques for standard ones in the simplex method. Bump-and-spike and merit techniques first are considered separately, then are briefly compared. Section 5 subsequently shows how staircase elimination techniques lead to additional efficiencies in the FTRAN and BTRAN routines.

4.1. *Bump-and-spike techniques*

The bump-and-spike methods originated by Hellerman and Rarick [28, 29] exemplify a 'global' approach to sparse elimination. They look for an overall permutation of B to a form that should have a naturally sparse LU factorization. An entire permutation is determined in advance of any numerical computations; during the numerical elimination stage, the permutation is modified only if an unacceptable pivot value is encountered.

As their name suggests, bump-and-spike techniques employ a two-stage procedure:

(1) The *bump-finding* phase determines an essentially unique permutation of B to a block-triangular form that has as many diagonal blocks ('bumps') as possible. (The block-triangular form somewhat resembles staircase form, but all its diagonal blocks B_{tt} are square and any sub-diagonal block may contain nonzero elements.)

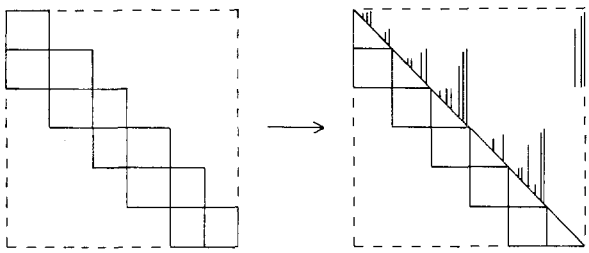
(2) The *spike-finding* phase tries to permute each block larger than 2×2 to a form that is entirely lower-triangular except for a small number of 'spike' columns that extend above the diagonal. This permutation is commonly entrusted to a heuristic algorithm known as P3 (the Preassigned Pivot Procedure).

Permuted in this way, B has a good structure for sparse Gaussian elimination. Creation of new nonzero elements in L and U—the 'fill-in'—is confined to the relatively few spike columns, while the remaining 'triangle' columns of B are essentially unaffected. Furthermore, elimination of a given spike column can cause fill-in only within subsequent columns of the same bump.

A proposed staircase bump-and-spike technique [17] dispenses with step (1) above by substituting a known staircase form for the block-triangular form. Step (2) then applies the P3 spike-finding heuristic to the diagonal blocks B_{tt} of the staircase, with appropriate modifications to handle blocks that are not square or that are rank-deficient. Rows of period 1 are thus assigned pivot elements first, followed by rows of period 2, period 3, and so forth through period T. The major difference is in the handling of columns. As explained in Section 2, a nonsingular staircase matrix always has at least as many columns as rows in the first t

periods; thus it is generally not possible to assign a pivot element from the block B_{tt} to every period-t column. Leftover columns from period t must be assigned pivot elements in rows of later periods; since these columns necessarily extend above the diagonal, they are referred to as 'interperiod spikes', in contrast to the 'intraperiod spikes' within B_{tt} that are found by P3.

The effect of this procedure (described in much more detail in [17]) is to permute the staircase so as to 'square off' its diagonal blocks while reducing it to a nearly lower-triangular form:



Fill-in is still confined to the spikes; elimination of a spike column can cause fill-in only within intraperiod spikes of the same block or within interperiod spikes of the same or preceding periods. The number of interperiod spikes is closely related to the cumulative excess of columns over rows, which is limited by the balance constraints of Section 2; thus there should be relatively few interperiod spikes and fill-in should be reasonably limited.

Computational experiments in [17] suggest that the standard and staircase bump-and-spike techniques are roughly competitive. They tend to produce comparable numbers of spikes, and the fill-in due to either technique is seldom much more than twice the fill-in due to the other. However, each technique does appear to be superior in certain situations.

Standard bump-and-spike seems invariably better when all bumps are small and most are 1×1. P3 is then applied cheaply to a few blocks, whereas the staircase technique must still apply P3 to every diagonal block of the staircase. The standard technique's spikes tend to be smaller than the staircase technique's interperiod spikes, and so the former fill in less; fill-in within L tends to be about the same, but the standard technique produces a notably sparser U. In addition, the standard technique produces fewer spikes that have unacceptable pivot values, and so wastes less time in modifying the chosen permutation.

Staircase bump-and-spike appears to have the advantage, however, when there are one or two very large block-triangular bumps that comprise a third or more of the rows and columns of B. Standard P3 is highly inefficient in processing these large bumps, whereas staircase P3 only needs to process the staircase blocks. In such situations the two techniques yield comparable fill-in within U, while the staircase technique yields a sparser L. Moreover, in some

cases the staircase technique produces substantially fewer spikes that have unacceptable pivots.

Both of these bump-and-spike techniques are designed for column-wise organization, and normally use the FACTOR-BY-COLUMN elimination routine of Section 3. Unacceptable pivot elements are handled by 'swapping' the pivot column with a later spike (an operation in which BTRANL figures as a subroutine). Storage for either technique need not exceed that required to hold B and the spike columns of L and U.

4.2. *Merit techniques*

'Merit' methods, as first proposed for linear programming by Markowitz [40], typify a 'local' approach to sparse elimination. They dynamically select the kth pivot element from within $\beta^{(k)}$ so as to guarantee relatively little fill-in of nonzero elements in computing $\beta^{(k+1)}$. For a sparse B, this myopic optimization of individual pivot elements tends to produce a sparse L and U overall, as computational experiments have confirmed [15].

Methods of this sort rely on a 'merit function that estimates—for each nonzero element of $\beta^{(k)}$—the fill-in that would result if that element were chosen as pivot. The kth pivot element is selected to minimize this merit function over all nonzero elements of $\beta^{(k)}$ that meet certain numerical requirements. Practical merit functions are generally computed from two simpler sets of values: $r_i^{(k)}$, the number of nonzeroes in row i of $\beta^{(k)}$, and $c_j^{(k)}$, the number of nonzeroes in column j of $\beta^{(k)}$. Markowitz originally suggested, for example, that the merit of $\beta_{ij}^{(k)}$ be computed as $(r_i^{(k)}-1)(c_j^{(k)}-1)$, which has proved both effective and efficient to implement [14, 15].

Proposed staircase merit techniques [17] restrict minimization of the merit function to roughly one period of $\beta^{(k)}$ at a time. As a consequence, both the rows and columns of B are assigned pivot elements in period order. Fill-in is thereby limited to a small part of $\beta^{(k)}$—roughly two periods or less—while the remainder of $\beta^{(k)}$ is just the same as B.

Staircase merit techniques should tend to produce a denser L and U than standard merit techniques, since the former minimize the merit function over a much smaller set of potential pivot elements. However, experiments in [17] suggest that the staircase techniques are not unreasonably inferior. In the worst test case the staircase merit technique produced about twice the fill-in (47% versus 22%); in some cases it did nearly as well as the standard technique, however, and in one case it was distinctly better.

Staircase merit techniques offer a clear economy, moreover, in carrying out the elimination. They confine all of the work of the kth elimination step—minimizing the merit function, computing $\beta^{(k+1)}$ from $\beta^{(k)}$, and updating $r_i^{(k)}$ and $c_j^{(k)}$—to the rows and columns of one or two periods. By contrast, the standard technique must deal with the entire $\beta^{(k)}$ at each elimination step.

Potential savings in storage are even greater. Merit techniques normally use the FACTOR-BY-ROW-AND-COLUMN algorithm of Section 3, since both rows and columns are dynamically permuted in selecting the pivot element. However, staircase merit elimination relies on only the one or two periods of BETA that differ from B, whereas standard merit elimination needs all of BETA. As a consequence, staircase merit techniques may be able to use simpler or more economical storage strategies than the standard techniques. As an example, under the standard techniques BETA shrinks only as L and U grow, and some sort of dynamic storage allocation is generally required to hold all three in available storage. Under the staircase techniques, by contrast, the active part of BETA stays small and fairly constant in size, and might well be kept in a fixed work area.

4.3. Comparison of bump-and-spike and merit techniques

There is no clear choice between bump-and-spike and merit techniques for sparse LU factorization, whether standard or staircase. Evidence of [17] suggests that each family of techniques offers the lowest fill-in for certain LP bases; additionally, each family is sensitive to the nature and availability of storage. To further complicate matters, particular LU updating schemes are designed for each family: for example, Saunders' scheme [53, 54] for bump-and-spike, Reid's [50, 51] for merit. These update schemes also have varying sparsity and storage characteristics.

Staircase bump-and-spike techniques do have one evident advantage: they apply just as well to higher-order staircase LPs (as defined in Section 2) as to first-order ones. Staircase merit techniques could also be adapted to handle higher-order staircases, but the extent of fill-in within $\beta^{(k)}$ would be greater and hence the savings over comparable standard techniques would be less.

On the other hand, staircase merit techniques are easily designed to ensure that all rows within a given square sub-staircase (Section 2) are assigned pivot elements on columns within that square sub-staircase. This 'respect' for sub-staircases—both lower and upper—may prove advantageous to BTRAN and FTRAN routines as discussed in Section 5. By contrast, staircase bump-and-spike techniques normally respect just sub-staircases of the lower-staircase structure; they can be made to respect upper sub-staircases only with some additional difficulty.

5. Solving staircase linear systems

The staircase elimination techniques discussed above have a significance to linear programming that goes beyond their different ways of computing $B = LU$. Both families of staircase techniques also make it possible to design specialized

BTRAN and FTRAN procedures for staircase LPs. These specialized procedures can contribute greatly to the efficiency of the simplex method, as the experiments in Section 6 will demonstrate.

For the purposes of BTRAN and FTRAN, the most important property of staircase elimination techniques is that they pass B's staircase structure on to L and U. Thus this section begins with a careful discussion of the period ordering of L and U. Thereafter each solution procedure—FTRANL, BTRANU, FTRANU, BTRANL—is taken up in turn.

5.1. Period partitions of L and U

Suppose that a factorization $PBQ^{\mathrm{T}} = LU$ has been determined as described in Section 3. In terms of this factorization and the staircase constraint matrix A, the following indices may be defined for each period $t = 1, \ldots, T$:

λ_t = first row of PB whose corresponding row of A is in period t or later.
μ_t = first column of BQ^{T} that is a column of A from period t or later.

Necessarily $\lambda_t \leq \lambda_{t+1}$ and $\mu_t \leq \mu_{t+1}$ for any choice of P, B and Q^{T}. Thus $\{\lambda_1, \ldots, \lambda_T\}$ and $\{\mu_1, \ldots, \mu_T\}$ partition the rows and columns, respectively, of PBQ^{T} by period. Since the rows of PBQ^{T} correspond to the rows of L, the λ_t values can also be thought of as partitioning L; analogously, the μ_t values partition U.

In general these partitions are not particularly significant, as the λ_t and μ_t values all tend to be small. In an extreme (but not unusual) case, for example, if the first row of PB is a period-T row, then $\lambda_1 = \cdots = \lambda_T = 1$.

If the factorization $PBQ^{\mathrm{T}} = LU$ is determined by one of the staircase elimination techniques, however, the λ_t and μ_t values must approximate the original staircase partitioning of B. This fact is clearest in the simple case where a staircase technique, from either family, is applied to the staircase structure that B inherits from A. These techniques all construct P so that PB preserves the staircase row ordering; hence they ensure $\lambda_t = \sum_1^{t-1} m_i + 1$. The merit techniques also construct Q so that BQ^{T} preserves the staircase column ordering, and consequently they also ensure $\mu_t = \sum_1^{t-1} n_j + 1$. For simple bump-and-spike techniques $\mu_t = \lambda_t$; more sophisticated versions give $\mu_t = \lambda_t +$ the number of all-zero rows in B_{tt}.

The situation is slightly more complicated if, as suggested in Section 2, B is put in reduced standard staircase form before the staircase elimination is carried out. Some rows of B that correspond to period-t rows of A may then be treated as if they are in period $t-1$. As a consequence, λ_t need only satisfy $\sum_1^{t-2} m_i + 1 \leq \lambda_t \leq \sum_1^{t-1} m_i + 1$. Thus a modification of the inherited staircase form to reduced standard form will tend to produce smaller λ_t values and a less regular λ-partition (although the μ-partition will be unaffected). The λ_t values will still be quite well-spaced, however, particularly if the periods are small and numerous.

A further complication is introduced when B's factorization is updated by the Bartels–Golub methods described in Section 3. P, L and hence the λ-partition are unchanged; Q and U are modified, however, and hence the μ-partition is altered. For example, the simplest update methods always delete one column of $B_{l-1}Q_{l-1}^{\mathrm{T}}$ and add a new column *at the end* to produce $B_lQ_l^{\mathrm{T}}$. All μ_t values greater than the index of the deleted column are consequently decreased by 1, and the μ-partition is thus slowly degraded. Nevertheless, degradation should not be severe for large LPs with the usual 50–100 updates between refactorizations. The situation is more involved in the case of sophisticated updates that add the new column *before the end* of B_{l-1}; the resulting degradation of the μ-partition should be no worse, however, provided some care is taken in choosing where to add the new column.

Staircase elimination techniques thus yield λ_t and μ_t values that significantly partition L and U by period. It remains to use these values to reduce the work of solving systems in L and U.

5.2. *Staircase FTRANL*

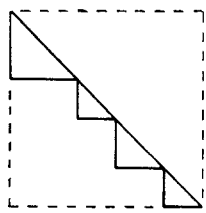
At each iteration FTRANL solves a system like $Lv = Pa$, where a is a column of A. If a is from period t, then it is zero on rows of periods 1 through $t-1$. Consequently,

$$(Pa)_i = 0 \quad \text{for all } i = 1, \ldots, \lambda_{t-1},$$

and so the main loop of FTRANL may begin at index λ_t as explained in Section 3.

In brief, when FTRANL transforms a period-t column it can start at period t in L, rather than at the beginning of L. The resultant savings should be modest, however, since FTRANL already handles zeroes efficiently.

Further savings may be possible when using an elimination technique that respects square sub-staircases in the upper-staircase structures (Section 4). If there are p upper sub-staircases, then L consists of p separate triangles:



The system $Lv = Pa$ thus decomposes into p independent triangular subsystems, one for each upper sub-staircase. Consequently, it is only necessary to solve the one or two triangular subsystems on whose rows a is nonzero; the other subsystems must have solutions that are all-zero. Savings through this scheme

would depend on the number of upper sub-staircases, and would probably be small since FTRANL is efficient to begin with; the extra logic involved in keeping track of the upper sub-staircases may sometimes be more trouble than it is worth.

5.3. Staircase BTRANU

At each iteration BTRANU solves a system like $U(Qy) = w$, where w is a solution vector produced by FTRANL plus update transformations. Generally nothing is known in advance about the position of zeroes in w even for staircase problems, although BTRANU may take advantage of individual zeroes in its usual efficient way.

A small saving is possible, however, if the location of (lower) square sub-staircases in B is known. Suppose, for example, that B has a sub-staircase at period t (that is, $\sum_1^t m_i = \sum_1^t n_i$). The system $By = a$ may therefore be permuted and partitioned as

$$\begin{bmatrix} B^{(11)} & 0 \\ B^{(21)} & B^{(22)} \end{bmatrix} \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix} = \begin{bmatrix} a^{(1)} \\ a^{(2)} \end{bmatrix}$$

where $B^{(11)}$ is the initial t-period square sub-staircase block. Suppose also that the column a is from any period $s > t$. By definition, its subvector $a^{(1)}$ is zero; hence necessarily $y^{(1)}$ is also zero.

In other words, under the above circumstances $y_j = 0$ whenever the jth column of B is from periods 1 through t. For any particular permutation Q, therefore,

$$(Qy)_j = 0 \quad \text{for all } j = 1, \ldots, \mu_t - 1.$$

Consequently the main loop of BTRANU, which computes $(Qy)_j$ in the order $j = m, \ldots, 1$, can stop after the μ_tth pass; the remainder of the solution is zero. The savings in this instance should be modest, since BTRANU handles zeroes efficiently in any case.

5.4. Staircase FTRANU

FTRANU solves a system like $U^T v = Qf$ at each iteration, where the vector f is commonly chosen in one of three ways:

(1) In phase 1 of the simplex method, f contains a ± 1 corresponding to each infeasible basic variable, and zeroes elsewhere.

(2) In phase 2, if the objective function is not included in B, then f is the vector of objective coefficients that correspond to the basic variables.

(3) In phase 2, if the objective function is included in B, then f contains one ± 1 and zeroes elsewhere.

In any of these cases, it may be possible to determine that f is zero in all

columns of the first t periods of the basis. It then follows that

$$(Qf)_j = 0 \quad \text{for all } j = 1, \ldots, \mu_t - 1,$$

and hence the main loop of FTRANU may begin at μ_t.

This situation is analogous to the one for FTRANL above: when FTRANU transforms an f that is zero prior to period t, it can start at period t in U rather than at the beginning of U. However, the *potential* savings are greater for FTRANU than for FTRANL (assuming U is stored in the usual column-wise fashion) because FTRANU cannot normally benefit from zeroes in f. The *actual* savings in FTRANU depend highly on how f is handled. Gains are most likely in case (1) above, where f tends to have many zeroes; savings in case (2) will be slight unless the objective has some special form. In case (3), FTRANU can be avoided entirely—even for non-staircase LPs—by placing the objective row last in PBQ^T; thus staircase FTRANU may have no work to save.

Staircase FTRANU might also take advantage of *lower* sub-staircases to avoid computing values that must be zero; the situation is analyzed just as for FTRANL above. Any savings would depend upon finding sub-staircases in which f is all-zero, and so the costs of finding such sub-staircases would have to be weighed against any benefits.

5.5. Staircase BTRANL

BTRANL produces a 'price vector', π, at each iteration by solving $L^T(P\pi) = w$, where w is the product of FTRANU and any update transformations. The simplex method then uses π to compute the reduced cost

$$d_j = c_j - \pi a_j$$

for any nonbasic variable x_j (with cost c_j and coefficient column a_j). If x_j is from period t, a_j must be all-zero except on the constraints of periods t and $t+1$; hence only the elements of π from these periods are needed to compute πa_j. Consequently only a portion of π may be needed at a given iteration if—by use of 'partial pricing'—only a subset of the d_j values are computed. (Numerous partial-pricing schemes appropriate to staircase LPs are discussed in [20].)

Suppose, therefore, that the current iteration needs only elements of π for periods t and later. Since $B^T\pi = f$, elements of π correspond in period to rows of B; elements of $P\pi$ thus correspond to rows of PB. To produce π for periods t and later, therefore, it suffices to compute

$$(P\pi)_i \quad \text{for all } i = \lambda_t, \ldots, m.$$

These are exactly the first $m - \lambda_t + 1$ elements produced, in reverse order, by BTRANL. Hence exactly the desired part of π is computed by running BTRANL until it completes the λ_t-pass of the main loop; the rest of BTRANL may be skipped. (Consult [20] for a more extensive discussion of this idea, including consideration of restarting BTRANL.)

Further savings might be achieved by trying to compute even smaller portions of π. Specifically, π might be computed within one upper-sub-staircase independently of the others. Such an arrangement would necessarily be complex, however, and its value is uncertain.

5.6. *Adaptations for higher-order staircases*

Most of the above methods are equally valid for higher-order staircases as described in Section 2, provided that some form of staircase Gaussian elimination is employed to assure good values of λ_t and μ_t. However, any savings due to upper-sub-staircases in FTRANL and BTRANL are likely to be lost.

6. Computational experience

This section reports initial computational experiments with some of the preceding ideas. The results indicate that staircase adaptation of the simplex method does make a significant difference: generally much less time is spent in certain routines, while more time is spent in others. Overall the staircase runs were measurably faster, and in one case the savings were quite substantial. Moreover, it appears there is significant room for improvement in subsequent implementations.

For the test runs an existing LP code, MINOS [44, 55], was modified to recognize staircase structure and to optionally apply some of the staircase techniques of Sections 4 and 5. Each test LP could then be solved twice—once with the staircase features turned off, once with them on—and the results could be meaningfully compared. Details of the test code and of the experimental setup are given in Appendix A.

MINOS employs a bump-and-spike factorization with Saunders' updating technique. Consequently a staircase bump-and-spike technique was implemented in the test version, and all test results bear directly only upon bump-and-spike methods. Nevertheless, from certain results one may make favorable speculations about the expected performance of staircase merit techniques, as indicated further below.

To keep the presentation compact, only short tables of results are presented here. Graphs of more extensive test data are collected in a technical report [18].

6.1. *Overall results*

Seven medium-to-large-scale linear programs were used in the tests. All are from applications, and are of dissimilar structures (aside from being staircase). Their gross dimensions (including objectives and right-hand sides) are given in Table 1. The 'iterations' column gives the number of iterations required by a standard version of MINOS to reach optimality from an all-slack starting basis.

Table 1

	Periods	Rows	Columns	Nonzero values	Iterations
SCAGR25	25	472	500	2208	1058
SCRS8	16	491	1169	4106	862
SCSD8	39	398	2750	11349	2047
SCFXM2	8	661	914	5466	1012
SCTAP2	10	1101	1880	13815	1174
PILOT	9	723	2789	9291	>2000
BP1	6	822	1571	11414	>2000

For the sake of economy, PILOT and BP1 were tested on runs of 1000 and 750 iterations, respectively, starting from advanced bases. The other problems were tested on runs to optimality from all-slack starts. Additional information about the test LPs is collected in Appendix B.

Raw results from the test runs, standardized to seconds per 1000 iterations, were as given in Table 2. Savings were substantial for PILOT, and respectable for SCSD8. For the others the gross difference between the standard and staircase techniques was small, though the latter performed worse only on BP1.

It is misleading to consider only these totals, however. When the times are broken down by function, it becomes evident that gains in some areas tend to be offset by losses in others. The staircase techniques have an edge in simplex pricing and pivoting, but are usually slightly behind in updating the LU factorization; they range from much faster to somewhat slower in pivot selection for Gaussian elimination, but are almost always slower in computing the L and U factors. Miscellaneous routines consume a good 10–20% of the time, much of which could be saved in practical (rather than test) circumstances.

Thus considerably more is to be learned by examining the times of individual routines and functions. The following subsections consider first the simplex-iteration routines, and then the LU-factorization ones.

Table 2

	Total time		
	Standard	Staircase	% change
SCAGR25	29.7	27.9	− 6%
SCRS8	33.9	31.5	− 7%
SCSD8	43.2	37.8	−13%
SCFXM2	43.4	42.2	− 3%
SCTAP2	67.2	67.1	0%
PILOT	155.7	106.4	−32%
BP1	181.8	189.7	+ 4%

6.2. Iterating routines

The simplex method spends a majority of its time in tasks that are repeated at each iteration: choosing a column to enter the basis (pricing), determining which column leaves the basis (pivoting), and revising the basis factorization accordingly (updating). The LP code's 'iterating' routines carry out these tasks.

For the test problems, total time spent in the iterating routines—again, normalized to seconds per thousand iterations—is given in Table 3. Here the results are somewhat more striking, with savings of 10–20% in four of the seven tests.

A further breakdown of these timings reveals that the greatest difference by far is in BTRANL, which is significantly faster with the staircase version in every instance. There is a corresponding, but smaller, efficiency in FTRANL. The normalized figures for these two routines are given in Table 4. Roughly the savings are 30–50% in BTRANL and 20–40% in FTRANL.

There is a small but noticeable tendency of the staircase version to run slower in BTRANU and FTRANU. Most likely this behavior is a consequence of the LU factorization: the staircase bump-and-spike elimination tends to yield a denser U.

Some of the difference in BTRAN and FTRAN timings should be due to the

Table 3

	Iterating time		
	Standard	Staircase	% change
SCAGR25	24.6	22.2	−10%
SCRS8	28.1	23.8	−15%
SCSD8	34.2	30.5	−11%
SCFXM2	33.2	32.5	− 2%
SCTAP2	56.9	54.3	− 5%
PILOT	108.0	86.3	−20%
BP1	136.6	146.1	+ 7%

Table 4

	FTRANL			BTRANL		
	Standard	Staircase	% change	Standard	Staircase	% change
SCAGR25	2.7	1.9	−29%	6.7	3.5	−48%
SCRS8	2.4	1.5	−36%	5.7	3.4	−41%
SCSD8	3.9	2.9	−25%	8.2	4.7	−42%
SCFXM2	2.6	1.9	−28%	7.8	5.4	−32%
SCTAP2	3.3	2.6	−21%	9.2	6.6	−28%
PILOT	13.0	8.0	−38%	22.9	12.7	−45%
BP1	14.8	12.6	−15%	32.5	26.9	−17%

Table 5

	Time saved by staircase FTRAN, BTRAN	% saving
SCAGR25	4.9	15%
SCRS8	4.1	12%
SCSD8	5.2	12%
SCFXM2	4.4	9%
SCTAP2	4.4	6%
PILOT	13.4	11%
BP1	3.5	2%

methods of Section 5. However, the efficiency of these methods cannot be told from the above data, which also reflect differing L and U densities. To get around this problem, a separate set of runs was made, employing the staircase LU factorization but not Section 5's enhancements. Table 5 gives the differences (per 1000 iterations) between runs with and without these enhancements: The efficiencies in FTRAN and BTRAN cut *total* running times 9–15% in most cases; the savings would be more pronounced as a percentage of iterating time only. Predictably, LPs of many periods tended to show the greatest differences.

Comparable savings should be realized if staircase bump-and-spike elimination techniques are replaced by staircase merit techniques, since the methods of Section 5 apply equally well to either. Hence merit techniques may well be superior for LPs such as SCAGR25 and SCFXM2 whose staircase factorizations—as reported in [17]—are notably denser under bump-and-spike.

The one sour note in the three tables above is BP1, for which the staircase iterating routines seem to perform rather poorly. On closer examination, however, this is not entirely surprising, as BP1 differs significantly from the other LPs. Whereas the others are first-order staircases (or, in the case of PILOT, very nearly first-order), BP1 has a large number of nonzeroes below the staircase; its form is in fact closer to dual-angular. BP1's bases consequently tend to be unbalanced. Hence the staircase technique produces considerably more spikes, and a much denser U factor. The result: much more time spent in FTRANU and BTRANU, offsetting any gains in FTRANL and BTRANL.

It thus appears that a good staircase form is essential to success of the staircase techniques. BP1's staircase araangement was deduced from fairly scant information, and is evidently inadequate. A better staircase form may exist, but but a better knowledge of the underlying model may be necessary to find it.

6.3. *Factorizing routines*

At intervals of typically 50–100 iterations a fresh factorization of the basis is computed by a separate set of routines. For bump-and-spike techniques, these

Table 6

	Factorizing time		
	Standard	Staircase	% change
SCAGR25	1.4	1.6	+15%
SCRS8	1.1	1.4	+22%
SCSD8	2.7	1.6	−39%
SCFXM2	1.9	2.8	+47%
SCTAP2	1.7	3.0	+80%
PILOT	32.8	9.7	−70%
BP1	27.9	26.1	− 6%

'factorizing' routines fall into two classes: ones that select a pivot order, and ones that compute the L and U factors.

For the test problems, total time in factorizing routines—normalized to seconds per 10 refactorizations—is given in Table 6. The outcomes appear to vary wildly. However, they are the consequence of a few simple patterns which are revealed by looking at the pivot-selection routines and LU-computation routines separately.

Pivot selection involves a routine for the P3 heuristic, a block-triangularization routine (for the standard technique only), and main routines to call these and record the selected pivots. The staircase technique's main routine seems to run usually somewhat longer, probably because it is more complicated. The other routines' normalized times are summarized in Table 7. The behavior of P3 is clearly critical. P3 is quite fast when bumps are small; but it begins to slow down when bump size passes 100, and it is extremely inefficient on bumps of size 400 or 500. PILOT, the worst case here, spends 16% of its total running time in P3 alone! By extrapolation, it seems likely that P3 will be prohibitively slow for large bumps. Thus a staircase bump-and-spike technique (or else an efficient merit technique) may be essential for larger versions of models like SCSD8 and PILOT.

The main LU computation routines employ FTRANL and BTRANL as

Table 7

	Standard			Staircase	
	P3	BLK Δ	Total	P3	Median size of largest bump
SCAGR25	0.4	0.2	0.6	0.2	45
SCRS8	0.2	0.2	0.4	0.2	28
SCSD8	1.1	0.4	1.5	0.2	114
SCFXM2	0.2	0.5	0.7	0.8	36
SCTAP2	0.0	0.5	0.5	0.7	1
PILOT	20.4	1.0	21.4	2.4	533
BP1	13.1	2.0	15.1	3.8	408

Table 8

	Standard *LU*				Staircase *LU*			
	Main	FTRANL	BTRANL	Swaps	Main	FTRANL	BTRANL	Swaps
SCAGR25	0.2	0.0	0.0	3	0.6	0.1	0.1	20
SCRS8	0.2	0.0	0.0	1	0.4	0.1	0.1	11
SCSD8	0.4	0.1	0.0	6	0.5	0.1	0.1	11
SCFXM2	0.5	0.1	0.0	2	1.0	0.2	0.1	8
SCTAP2	0.2	0.0	0.0	0	0.7	0.1	0.4	19
PILOT	3.3	3.8	2.4	27	3.2	1.8	0.8	16
BP1	3.7	3.8	2.4	28	6.4	5.8	7.2	49

subroutines. FTRANL solves for the next column of L and U (as described in Section 3), and BTRANL solves for row k of $\beta^{(k)}$ when a column interchange ('spike swap') is necessitated by an unacceptable pivot element. The test problems gave the normalized results of Table 8 (where 'swaps' is the maximum number of swapped spikes per factorization). These times are sensitive to the numbers of spike swaps, since each swap requires another BTRANL and FTRANL plus extra work in the main routine. Experience with PILOT and other LPs [17] suggests that the staircase technique may generally require fewer swaps when the block-triangular bumps are big (as for PILOT) and the staircase is well-balanced (unlike BP1's). However, the staircase technique seems to need more swaps when the block-triangular form has relatively small bumps.

Again the data suggest that staircase merit techniques might be preferable for the small-bump staircase LPs. An efficient implementation of merit-function minimization [14, 51] need incur only a small extra cost in rejecting any unacceptably small pivot element.

6.4. *Comparison with a commercial code*

Production runs of the PILOT model were frequently made on the same computer as used for the above tests. These runs employed a commercially-marketed machine-language LP code—the WHIZARD simplex routine of MPS III [43]—which incorporated a bump-and-spike factorization scheme. Various system parameters were set from experience to yield fast PILOT runs.

For comparison, therefore, WHIZARD was run 1000 iterations from the same starting basis as used above with MINOS. The running times were as follows:

MINOS, standard 155.7 sec,

MPS III/WHIZARD 114.7 sec,

MINOS, staircase 106.4 sec.

MINOS did require considerably more storage, primarily because its storage scheme for the U factor could not efficiently accommodate a large number of

spikes. U could probably be stored more compactly, however, without significant effect upon the MINOS timings.

Nothing very definite can be inferred from these figures, since MINOS and MPS III-WHIZARD differ in many ways. They employ different scaling techniques, factorization routines, and refactorization frequencies; moreover, WHIZARD uses multiple pricing while MINOS does not. Nevertheless, it is encouraging that staircase MINOS can be compared at all with a fast commercial LP system. At the least, one may conclude that the timings throughout this section are probably fairly realistic.

7. Conclusions

The preceding experimental evidence clearly bears out one hypothesis of this paper: that the inversion routines of the simplex method may be adapted to handle staircase LPs more efficiently. Are these staircase efficiencies sufficiently general and substantial to be of practical significance? There is reason to believe that they are, but a conclusive answer must await two further sorts of evidence.

First, it must be determined whether all inversion routines can be improved together by staircase adaptations, or whether improvements in some routines can merely be traded off against degradations of others. Several trade-offs are evident in the experiments of Section 6; most seriously, staircase efficiencies in FTRAN and BTRAN routines require staircase sparse-elimination techniques that are sometimes slower and yield a denser factorization (than standard techniques). This trade-off may well be eliminated, however—as suggested previously—by taking a fresh look at the merit elimination techniques, which have been overshadowed by the bump-and-spike techniques in simplex-method implementations. Staircase merit techniques have neither the great storage requirements of standard merit techniques nor the large-spike problems of staircase bump-and-spike techniques; hence they may permit efficiencies in solution routines without introducing great inefficiencies into the factorization routines.

Second, it will be important to determine whether staircase savings grow with increased LP size and difficulty. Section 6's experiments suggest a favorable trend, in which staircase savings tend to be least for the small and 'easy' LPs and to be greatest for large and 'hard' ones that are most expensive to begin with. This trend is particularly clear in certain routines; for example, the staircase P3 heuristic is highly advantageous for difficult bases with large block-triangular bumps, and the staircase BTRAN and FTRAN routines have the greatest edge when the number of periods is large.

It thus seems reasonable to hope to achieve the broader goal of this paper: solving staircase linear programs at meaningfully lower cost. Experiments here have shown the possibility of savings in the inversion routines alone; the

companion paper [20] will show a potential for equally great savings through specialized staircase pricing techniques.

Acknowledgments

Thanks are due to Professor George Dantzig for supporting this research and to Dr. Michael Saunders for his numerous suggestions and his close reading of an early draft.

This research was carried out at the Systems Optimization Laboratory of the Department of Operations Research, Stanford University, and at the Stanford Office of the National Bureau of Economic Research. Financial support was provided in part through the Systems Optimization Laboratory by Department of Energy contract DE-AS03-76-SF00326 PA# DE-AT03-76-ER72018, by Office of Naval Research contract N00014-75-C-0267, and by National Science Foundation grants MCS76-20019 A01 and ENG77-06761.

Appendix A: Details of computational tests

A.1. Computing environment

All computational experiments were performed on the Triplex system [57] at the Stanford Linear Accelerator Center, Stanford University. The Triplex comprised three computers linked together: one IBM 360/91, and two IBM 370/168s. Runs were submitted as batch jobs in a virtual-machine environment, under the control of IBM systems OS/VS2, OS/MVT and ASP.

Test runs employed a specially-modified set of linear-programming routines from the MINOS system [44, 55]. MINOS is written in standard FORTRAN. For timed runs, MINOS was complied with the IBM FORTRAN IV (H extended, enhanced) compiler, version 1.1.0, at optimization level 3 [36].

A.2. Timings

All running-time statistics are based on 'CPU second' totals for individual job steps as reported by the operating system. To promote consistency all timed jobs were run on the Triplex computer designated 'system A', and jobs whose timings would be compared were run at about the same time. Informal experiments indicated roughly a 1% variation in timings due to varying system loads.

More detailed timings employed PROGLOOK [37], which takes frequent samples of a running program to estimate the proportion of time spent in each subroutine. To determine the actual time in seconds for each subroutine, every timed job was run twice—once without PROGLOOK to measure total CPU

seconds, and once with PROGLOOK to estimate each subroutine's proportion of the total. PROGLOOK estimates were based on at least 2300 samples per job.

A.3. MINOS linear-programming evnironment

MINOS was set up for test runs according to the defaults indicated in [44], with the exception of the items listed below.

Scaling. All test runs of SCRS8, SCFXM2, PILOT and BP1 employed scaled versions of these problems. In every case the scaling was determined by the following geometric-mean procedure (in which A denotes the coefficient matrix exclusive of objective and right-hand side):

SET ρ_0 = maximum ratio of magnitudes of any two nonzero elements in the same column of A.
REPEAT for $k = 1, 2, 3, \ldots$:
  DIVIDE each row i of A, and its corresponding right-hand-side value, by $[(\min_j |A_{ij}|)(\max_j |A_{ij}|)]^{1/2}$, taking the minimum over all nonzero elements in row i.
  DIVIDE each column j of A, and its corresponding coefficient in the objective, by $[(\min_i |A_{ij}|)(\max_i |A_{ij}|)]^{1/2}$, taking the minimum over all nonzero elements in column j.
  SET ρ_k = maximum ratio of magnitudes of any two nonzero elements in the same column of A, as scaled.
UNTIL $\rho_k \geq (0.9)\rho_{k-1}$.

The maximum-column-ratio criterion, ρ_k, was employed because MINOS uses a related criterion to determine the acceptability of pivot values in LU factorization.

Starting basis. All LPs except PILOT and BP1 were solved with crash option 0 of MINOS; the initial basis was composed entirely of unit vectors, and all nonbasic variables were placed at zero. PILOT and BP1 were run from initial bases that had been reached and saved in previous MINOS runs.

Termination. All LPs except PILOT and BP1 were run until an optimal solution was found. PILOT and BP1 were run for 1000 and 750 iterations, respectively.

Pricing. Except for SCTAP2, the partial-pricing scheme of MINOS was employed, with one important change: the arbitrary partitioning of the columns normally defined by MINOS for partial pricing was replaced by the natural staircase partitioning. Thus the periods of the staircase were priced one at a time in a cyclic fashion.

Pricing for SCTAP2 was similar except that the incoming column was chosen from the latest possible period. (This choice was known to produce a relatively small number of iterations from an all-unit-vector start.)

Refactorization frequency. MINOS was instructed to refactorize the basis (by

performing a fresh Gaussian elimination) every 50 iterations, except for BP1 (every 75) and PILOT (every 90).

Tolerances. The '*LU* ROW TOL' for MINOS was set to 10^{-4}. All other tolerances were left at their default values.

A.4. Modifications to MINOS

All runs described in this paper were made with a special test version of MINOS. This version retained MINOS' routines for standard bump-and-spike elimination, and added new routines to implement a version of staircase bump-and-spike elimination. Routines for solving linear systems were also modified to take advantage of the staircase pivot order. Control routines were adjusted appropriately.

New subroutines in the test version are described briefly as follows:

SP3—an adaptation of the P3 heuristic to find a bump-and-spike structure in non-square or rank-deficient blocks, as proposed in [17]. This routine is a modification of the MINOS subroutine P3.

SP4—main routine for the staircase bump-and-spike pivot-selection technique of [17]; sorts the staircase basis into reduced form, and calls SP3 once for each staircase diagonal block.

DSPSPK—spike-display routine; prints a graphical summary of the basis bump-and-spike structure found by P4 (for the standard technique) or SP4 (for the staircase technique).

STAIR—a staircase analyzer. Given an initial partition of the rows by period, this routine permutes the constraint matrix to a reduced standard staircase form and stores the staircase partitions in arrays that are read by subsequent routines. STAIR is called once at the beginning of every run.

SCALE—implementation of the geometric-mean scaling scheme described above; called optionally at the beginning of a run.

UPDBAL—updating routine for cumulative-balance counts: after each iteration, revises an array that records the cumulative excess of columns over rows at each period of the staircase basis. (This array is used to find square sub-staircases.)

In addition the test version incorporates the following substantial modifications to MINOS subroutines:

FACTOR efficiently handles a pivot order from either the standard or staircase technique, and finds the partitions λ_t and μ_t (defined in Section 5) for the staircase technique.

FTRANL, BTRANL, FTRANU and BTRANU incorporate ideas of Section 5 in a uniform way. FTRANL and FTRANU can begin at a specified L or U transformation, and BTRANL and BTRANU can stop at a specified transformation. BTRANL can also be restarted at a point where it previously stopped.

LPITN determines a starting point for FTRANL and a stopping point for BTRANU when the staircase technique is used. (No attempt is made to prematurely stop FTRANL, however.)

SETPI, for the staircase technique, determines a starting point for FTRANU and a stopping point for BTRANL when it is first called at an iteration. When subsequently called at the same iteration it determines restarting and stopping points for BTRANL. (No attempt is made to prematurely stop FTRANU, however, or to stop or start BTRANL based on square sub-staircases.)

PRICE incorporates the staircase-oriented partial-pricing methods described in Section A.3. When these methods are used with the staircase factorization technique, PRICE also keeps track of how much of the price vector it requires, and calls SETPI accordingly.

SPECS2 determines whether the standard or staircase technique will be used in a particular run, according to instructions in the SPECS input file.

Other subroutines were modified as necessary to accommodate these changes.

A.5. MPS III linear programming environment

For purposes of comparison the PILOT test problem was also run on the MPS III system [43], as explained in Section 6.

The MPS III run employed the WHIZARD linear-programming routines of version 8915 of MPS III. The run used the same starting basis as the MINOS runs for PILOT, and was terminated after 1000 iterations like the MINOS runs. Exact CPU timings were 0.56 seconds in the compiler step and 114.18 seconds in the executor step.

The control program for the MPS III run was as follows:

```
            PROGRAM
            INITIALZ
            XPROC = XPROC + 6000
            XCLOCKSW = 0
            XINVERT = 1
            XFREQINV = 90
            XFREQLGO = 1
            XFREQ1 = 1000
            MVADR (XDOFREQ1, TIME)
            MOVE (XDATA, 'PILOT.WE')
            CONVERT ('FILE', 'INPUT')
            SETUP ('BOUND', 'BOUND', 'MAX', 'SCALE')
            MOVE (XOBJ, 'OBJ')
            MOVE (XRHS, 'RHSIDE')
            INSERT ('FILE', 'PUNCH1')
WHIZFREQ    DC (250)
```

```
WHIZCAL   DC (4)
          WHIZARD ('FREQ', WHIZFREQ, 'SCALE', WHIZSCAL)
TIME      PUNCH ('FILE', 'PUNCH1')
          EXIT
          PEND
```

Appendix B: Test problems

The linear programs used in the computational experiments of Section 6 are described in greater detail below. Comparative summaries and statistics appear first, followed by detailed statistical descriptions of the LPs' staircase structures.

All of these linear programs are available from the author in computer-readable MPS format, on cards or tape. SCAGR25, SCRS8, SCSD8, SCFXM2 and SCTAP2 are also available as part of a larger set of staircase LPs distributed by Ho and Loute [34].

B.1. Origins of the test LPs

SCAGR25 is a planning model for expansion of a large dairy farm, developed by Swart, Smith and Holderby [56].

SCRS8 is derived from a model of the United States' options for a transition from oil and gas to synthetic fuels. It was constructed by Ho [31] based on a model by Manne [39].

SCSD8 models the minimal-weight design of multi-stage trusses under a single loading condition, as described by Ho [30]. This is the only staircase test problem (for this paper) in which the stages do not represent periods of time.

SCFXM2 is described by Ho and Loute [34] as an extension of a real-world problem in production scheduling.

SCTAP2 optimizes the dynamic flow over a traffic network in which congestion is modeled explicitly by the flow equations. This LP was formulated by Merchant and Nemhauser [42] and further studied by Ho [32]. (The LP is distributed with 11 potential objective rows; the objective named OBJZZZZZ was used in all tests for this paper. All statistics below omit the other ten objectives.)

PILOT is derived from a welfare equilibrium model of the United States' energy supply, energy demand, and economic growth, documented by Parikh [46]. The LP was supplied by the Systems Optimization Laboratory of the Department of Operations Research, Stanford University.

BP1 was developed by British Petroleum, London; the details of its origins are unknown to the present author. The LP was supplied via the Systems Optimization Laboratory of the Department of Operations Research, Stanford University. (The structure of this LP is approximately dual-angular, with 6 main diagonal

blocks and about 400 coupling variables. For the experiments described in this paper it was treated as a 6-period staircase problem with some elements below the first-order staircase.)

B.2. Summary statistics

Tables 9 and 10 describe the matrix A of constraint coefficients for each test problem, exclusive of any objective or right-hand-side vectors. Thus the numbers of constraints and nonzero coefficients are somewhat smaller than the values given in Section 6. The 'density' is the proportion of nonzero elements in A.

The 'unscaled' values refer to the test problems as originally received; the 'scaled' values were computed after application of the scaling procedure described in Appendix A. 'Max elem' and 'min elem' are the largest and smallest magnitudes, respectively, among the nonzero elements of A. The 'largest col

Table 9

	Constraints			Variables[a]	Nonzero coefficients	Density
	Eq	Ineq	Total			
SCAGR25	300	171	471	500	1554	0.66%
SCRS8	384	106	490	1169	3182	0.56%
SCSD8	397	—	397	2750	8584	0.79%
SCFXM2	374	286	660	914	5183	0.86%
SCTAP2	470	620	1090	1880	6714	0.33%
PILOT	583	139	722	2789	9126	0.45%
BP1	516	305	821	1571	10400	0.81%

[a]PILOT has 80 free variables, 296 upper-bounded variables, and 79 fixed variables. Otherwise, all variables in all problems are required only to be nonnegative.

Table 10

	Unscaled			Scaled		
	Max elem	Min elem	Largest col ratio	Max elem	Min elem	Largest col ratio
SCAGR25[a]	9.3	$2.0 \cdot 10^{-1}$	$1.9 \cdot 10^{1}$	—	—	—
SCRS8	$3.9 \cdot 10^{2}$	$1.0 \cdot 10^{-3}$	$4.5 \cdot 10^{3}$	4.0	$2.5 \cdot 10^{-1}$	$1.6 \cdot 10^{1}$
SCSD8[a]	1.0	$2.4 \cdot 10^{-1}$	4.0	—	—	—
SCFXM2	$1.3 \cdot 10^{2}$	$5.0 \cdot 10^{-4}$	$1.3 \cdot 10^{5}$	$1.1 \cdot 10^{1}$	$8.7 \cdot 10^{-2}$	$1.3 \cdot 10^{2}$
SCTAP2[a]	$8.0 \cdot 10^{1}$	1.0	$8.0 \cdot 10^{1}$	—	—	—
PILOT	$4.8 \cdot 10^{4}$	$1.4 \cdot 10^{-4}$	$7.0 \cdot 10^{6}$	$2.0 \cdot 10^{1}$	$4.9 \cdot 10^{-2}$	$4.2 \cdot 10^{2}$
BP1	$2.4 \cdot 10^{2}$	$2.0 \cdot 10^{-4}$	$1.7 \cdot 10^{5}$	$1.3 \cdot 10^{1}$	$7.6 \cdot 10^{-2}$	$1.7 \cdot 10^{2}$

[a]Not scaled prior to test runs.

ratio' is the greatest ratio of any two nonzero magnitudes in the same column of A.

B.3. Staircase-structure statistics

Tables 11–17 describe the staircase structures of the individual test problems. In each table the line for period t refers to the objective and constraint coefficients for the period-t variables; where successive periods are identical in structure their entries have been combined.

Table 11
SCAGR25

	Diagonal blocks				Off-diagonal blocks				
Period	Rows	Cols.	Elems.	Dens.	Rows	Cols.	Elems.	Dens.	Obj. elems.
1	18	20	45	13%	8	7	17	30%	19
2–24	19	20	46	12%	8	7	17	30%	19
25	16	20	43	13%					19
			1146	12%			408	30%	475

Table 12
SCRS8

	Diagonal blocks				Off-diagonal blocks				
Period	Rows	Cols.	Elems.	Dens.	Rows	Cols.	Elems.	Dens.	Obj. elems.
1	28	37	65	6%	25	22	29	5%	18
2	28	38	69	6%	25	22	29	5%	19
3–5	31	76	181	8%	25	22	29	5%	55
6–8	32	79	192	8%	25	22	29	5%	58
9	31	79	189	8%	25	22	29	5%	58
10–12	31	80	190	8%	25	22	29	5%	59
13–15	30	80	186	8%	25	22	29	5%	59
16	31	70	177	8%					59
			2747	8%			435	5%	847

Table 13
SCSD8

	Diagonal blocks				Off-diagonal blocks				
Period	Rows	Cols.	Elems.	Dens.	Rows	Cols.	Elems.	Dens.	Obj. elems.
1–38	10	70	130	19%	10	50	90	18%	70
39	17	90	224	15%					90
			5164	18%			3420	18%	2750

Table 14
SCFXM2

	Diagonal blocks				Off-diagonal blocks				
Period	Rows	Cols.	Elems.	Dens.	Rows	Cols.	Elems.	Dens.	Obj. elems.
1	92	114	679	6%	9	57	61	12%	13
2	82	99	434	5%	9	35	35	11%	4
3	66	126	300	4%	5	33	33	20%	1
4	90	118	1047	10%	5	5	5	20%	5
5	92	114	679	6%	9	57	61	12%	13
6	82	99	434	5%	9	35	35	11%	4
7	66	126	300	4%	5	33	33	20%	1
8	90	118	1047	10%					5
			4920	7%			263	13%	46

Table 15
SCTAP2

	Diagonal blocks				Off-diagonal blocks				
Period	Rows	Cols.	Elems.	Dens.	Rows	Cols.	Elems.	Dens.	Obj. elems.
1–9	109	188	423	2%	62	138	276	3%	141
10	109	188	423	2%					141
			4230	2%			2484	3%	1410

Table 16
PILOT

	Diagonal blocks				Off-diagonal blocks				Sub-stair blocks		Obj. elems.
Period	Rows	Cols.	Elems.	Dens.	Rows	Cols.	Elems.	Dens.	Elems.	Dens.	
1	84	343	686	2%	31	74	105	5%	18	0%	10
2	90	345	1079	3%	34	76	111	4%	8	0%	10
3	90	343	1073	3%	34	74	109	4%	5	0%	10
4	90	343	1073	3%	34	74	109	4%	5	0%	10
5	90	343	1073	3%	34	74	109	4%	5	0%	10
6	90	343	1073	3%	34	74	109	4%	3	0%	10
7	90	343	1073	3%	32	74	107	5%	1	0%	10
8	87	341	1060	4%	4	19	19	25%			10
9	11	45	113	23%							12
			8303	3%			778	4%	45	0%	92

Table 17
BP1

Period	Diagonal blocks				Off-diagonal blocks				Sub-stair blocks		Obj. elems.
	Rows	Cols.	Elems.	Dens.	Rows	Cols.	Elems.	Dens.	Elems.	Dens.	
1	111	227	1400	6%	3	60	3	2%	163	0%	138
2	151	353	2175	4%	62	108	112	2%	142	0%	149
3	113	321	964	3%	92	232	346	2%	494	1%	270
4	170	295	2178	4%	51	14	11	2%	4	0%	74
5	134	198	1315	5%	111	2	2	1%			40
6	142	177	1091	4%							56
			9123	4%			474	2%	803	0%	727

In each case the constraint matrix A has been put in reduced standard form as described in Section 2. 'Diagonal blocks' refers to the staircase blocks A_{tt}, 'off-diagonal blocks' to the blocks $\hat{A}_{t+1,t}$, and 'sub-stair blocks' (when present) to the blocks $A_{t+2,t}, \ldots, A_{T,t}$. The given densities are the percentages of nonzero elements in the relevant blocks.

References

[1] T. Aonuma, "A two-level algorithm for two-stage linear programs", *Journal of the Operations Research Society of Japan* 21 (1978) 171–187.

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

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

[4] M. Benichou et al., "The efficient solution of large-scale linear programming problems—Some algorithmic techniques and computational results", *Mathematical Programming* 13 (1977) 280–322.

[5] J. Bisschop and A. Meeraus, "Matrix augmentation and structure preservation in linearly constrained control problems", *Mathematical Programming* 18 (1980) 7–15.

[6] R.H. Cobb and J. Cord, "Decomposition approaches for solving linked problems", in: H.W. Kuhn, ed., *Proceedings of the Princeton symposium on mathematical programming* (Princeton University Press, Princeton, NJ, 1970) pp. 37–49.

[7] G.B. Dantzig, "Programming of interdependent activities, II: mathematical model", *Econometrica* 17 (1949) 200–211.

[8] G.B. Dantzig, "Upper bounds, secondary constraints and block triangularity in linear programming", *Econometrica* 23 (1955) 174–183.

[9] G.B. Dantzig, "Optimal solution of a dynamic Leontief model with substitution", *Econometrica* 23 (1955) 295–302.

[10] G.B. Dantzig, "Compact basis triangularization for the simplex method", in: R.L. Graves and P. Wolfe, eds., *Recent advances in mathematical programming* (McGraw-Hill, New York, 1963) pp. 125–132.

[11] G.B. Dantzig, "Solving staircase linear programs by a nested block-angular method", Technical Report 73-1, Department of Operations Research, Stanford University, Stanford, CA (1973).

[12] G.B. Dantzig and P. Wolfe, "Decomposition principle for linear programs", *Operations Research* 8 (1960) 101–111.

[13] I.S. Duff, "On the number of nonzeroes added when Gaussian elimination is performed on sparse random matrices", *Mathematics of Computation* 28 (1974) 219–230.
[14] I.S. Duff, "Practical comparisons of codes for the solution of sparse linear systems", in: I.S. Duff and G.W. Stewart, eds., *Sparse matrix proceedings* 1978 (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979) pp. 107–134.
[15] I.S. Duff and J.K. Reid, "A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination", *Journal of the Institute of Mathematics and its Applications* 14 (1974) 281–291.
[16] J.J.H. Forrest and J.A. Tomlin, "Updated triangular factors of the basis to maintain sparsity in the product form simplex method", *Mathematical Programming* 2 (1972) 263–278.
[17] R. Fourer, "Sparse Gaussian elimination of staircase linear systems", Technical Report SOL 79–17, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA (1979).
[18] R. Fourer, "Solving staircase linear programs by the simplex method, 1: inversion", Technical Report SOL 79–18, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA (1979); reprinted in: G.B. Dantzig, M.A.H. Dempster and M.J. Kallio eds., *Large-scale linear programming* 1 (International Institute for Applied Systems Analysis, Laxenburg, 1981) pp. 179–259.
[19] R. Fourer, "Staircase matrices", Technical Report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL (1980).
[20] R. Fourer, "Solving staircase linear programs by the simplex method, 2: pricing", *Mathematical Programming* 24 (1982) to appear.
[21] D.M. Gay, "On combining the schemes of Reid and Saunders for sparse LP bases", in: I.S. Duff and G.W. Stewart, eds., *Sparse matrix proceedings* 1978 (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979) pp. 313–334.
[22] C.W. Gear et al., "Numerical computation: its nature and research directions", *SIGNUM Newsletter* (Association for Computing Machinery, New York, 1979).
[23] C.R. Glassey, "Dynamic linear programs for production scheduling", *Operations Research* 19 (1971) 45–56.
[24] C.R. Glassey, "Nested decomposition and multi-stage linear programs", *Management Science* 20 (1973) 282–292.
[25] D. Goldfarb, "On the Bartels–Golub decomposition for linear programming bases", *Mathematical Programming* 13 (1977) 292–292.
[26] R.C. Grinold, "Steepest ascent for large-scale linear programs", *SIAM Review* 14 (1972) 447–464.
[27] A.R.G. Heesterman and J. Sandee, "Special simplex algorithm for linked problems", *Management Science* 11 (1965) 420–428.
[28] E. Hellerman and D. Rarick, "Reinversion with the preassigned pivot procedure", *Mathematical Programming* 1 (1971) 195–216.
[29] E. Hellerman and D.C. Rarick, "The partitioned preassigned pivot procedure (P^4)", in: D.J. Rose and R.A. Willoughby, eds., *Sparse matrices and their applications* (Plenum Press, New York, 1972) pp. 67–76.
[30] J.K. Ho, "Optimal design of multi-stage structures: a nested decomposition approach", *Computers and Structures* 5 (1975) 249–255.
[31] J.K. Ho, "Nested decomposition of a dynamic energy model", *Management Science* 23 (1977) 1022–1026.
[32] J.K. Ho, "A successive linear optimization approach to the dynamic traffic assignment problem", *Transportation Science* 14 (1980) 295–305.
[33] J.K. Ho and E. Loute, "A comparative study of two methods for staircase linear programs", *ACM Transactions on Mathematical Software* 6 (1980) 17–30.
[34] J.K. Ho and E. Loute, "A set of staircase linear programming test problems", *Mathematical Programming* 20 (1981) 245–250.
[35] J.K. Ho and A.S. Manne, "Nested decomposition for dynamic models", *Mathematical Programming* 6 (1974) 121–140.
[36] "IBM OS FORTRAN IV (H extended) compiler programmer's guide", No. SC28-6852, International Business Machines Corporation (New York, 1974).

[37] R. Johnson and T. Johnston, "PROGLOOK user's guide", User Note 33, SLAC Computing Services, Stanford Linear Accelerator Center, Stanford, CA (1976).
[38] O.B.G. Madsen, "Solution of LP-problems with staircase structure", Research Report 26, The Institute of Mathematical Statistics, Lyngby (1977).
[39] A.S. Manne, "U.S. options for a transition from oil and gas to synthetic fuels", Discussion Paper 26D, Public Policy Program, Kennedy School of Government, Harvard University, Cambridge, MA (1975).
[40] H.M. Markowitz, "The elimination form of the inverse and its application to linear programming", *Management Science* 3 (1957) 255–269.
[41] R.E. Marsten and F. Shepardson, "A double basis simplex method for linear programs with complicating variables", Technical Report 531, Department of Management Information Systems, University of Arizona, Tucson, AZ (1978).
[42] D.K. Merchant and G.L. Nemhauser, "A model and an algorithm for the dynamic traffic assignment problems", *Transportation Science* 12 (1978) 183–199.
[43] "MPS III mathematical programming system: user manual", Ketron Inc., Arlington, VA (1975).
[44] B.A. Murtagh and M.A. Saunders, "MINOS: a large-scale nonlinear programming system (for problems with linear constraints): user's guide", Technical Report SOL 77-9, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA (1977).
[45] W. Orchard-Hays, *Advanced linear-programming computing techniques* (McGraw-Hill, New York, 1968).
[46] S.C. Parikh, "A welfare equilibrium model (WEM) of energy supply, energy demand, and economic growth", Technical Report SOL 79-3, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA (1979).
[47] A.F. Perold, "Fundamentals of a continuous time simplex method", Technical Report SOL 78-26, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA (1978).
[48] A.F. Perold and G.B. Dantzig, "A basis factorization method for block triangular linear programs", in: I.S. Duff and G.W. Stewart, eds., *Sparse matrix proceedings* 1978 (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979) pp. 283–312.
[49] A. Propoi and V. Krivonozhko, "The simplex method for dynamic linear programs", Report RR-78-14, International Institute for Applied Systems Analysis, Laxenburg (1978).
[50] J.K. Reid, "A sparsity-exploiting variant of the Bartels–Golub decomposition for linear programming bases", Report CSS 20, Computer Science and Systems Division, A.E.R.E., Harwell (1975).
[51] J.K. Reid, "Fortran subroutines for handling sparse linear programming bases", Report AERE-R8269, Computer Science and Systems Division, A.E.R.E., Harwell (1976).
[52] R. Saigal, "Block-triangularization of multi-stage linear programs", Report ORC 66-9, Operations Research Center, University of California, Berkeley, CA (1966).
[53] M.A. Saunders, "The complexity of *LU* updating in the simplex method", in: R.S. Anderssen and R.P. Brent, eds., *The complexity of computational problem solving* (Queensland University Press, Brisbane, Qld., 1975) pp. 214–230.
[54] M.A. Saunders, "A fast, stable implementation of the simplex method using Bartels–Golub updating", in: J.R. Bunch and D.J. Rose, eds., *Sparse matrix computations* (Academic Press, New York, 1976) pp. 213–226.
[55] M.A. Saunders, "MINOS system manual", Technical Report SOL 77-31, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA (1977).
[56] W. Swart, C. Smith and T. Holderby, "Expansion planning for a large dairy farm", in: H.M. Salkin and J. Saha, eds., *Studies in linear programming* (American Elsevier, New York, 1975) pp. 163–182.
[57] I. Vinson, "Triplex user's guide", User Note 99, SLAC Computing Services, Stanford Linear Accelerator Center, Stanford, CA (1978).
[58] R.D. Wollmer, "A substitute inverse for the basis of a staircase structure linear program", *Mathematics of Operations Research* 2 (1977) 230–239.